Security Constraints on the Oswald-Aigner Exponentiation Algorithm

Colin D. Walter

Comodo Research Lab
10 Hey Street, Bradford, BD7 1DQ, UK
Colin.Walter@comodogroup.com www.comodogroup.com

Abstract. In smartcard encryption and signature applications, randomized algorithms can be used to increase tamper resistance against attacks based on averaging data-dependent power or EMR variations. Recently, Oswald and Aigner described such an algorithm suitable for point multiplication in elliptic curve cryptography (ECC). With the assumption that an attacker can identify additions and doublings and distinguish them from each other during a single point multiplication, it is shown that the algorithm is insecure for repeated use of the same secret key without blinding of that key. This scotches hopes that the expense of such blinding might be avoided by using the algorithm unless the differences between point additions and doublings can be obscured successfully.

Key words: Addition-subtraction chains, randomized exponentiation, elliptic curve cryptography, ECC, point multiplication, power analysis, SPA, DPA, SEMA, DEMA, blinding, smartcard.

1 Introduction

Major progress in the theory and practice of side channel attacks [6, 7] on embedded cryptographic systems shows that substantial data about secret keys can leak from a *single* application of a cryptographic function through data-dependent power variation and electro-magnetic radiation [12, 13]. This is particularly true for the more computationally intensive functions such as exponentiation, which is a major process in many crypto-systems such as RSA, ECC and Diffie-Hellman. Initial attacks of this type required averaging over a number of exponentiations [9] to extract meaningful data, but improved techniques mean that single exponentiations using traditional algorithms are no longer safe. In particular, it should be assumed that the pattern of squares and multiplies can be extracted fairly accurately from side channel leakage. If the standard binary "square-and-multiply" algorithm is used, this pattern reveals the secret exponent immediately. More generally, operand reuse might be determined as well and this used to extract the secret key when sliding windows are employed [16]. Deterministic re-coding does little to improve matters [11].

In this context, Oswald and Aigner proposed a randomized point multiplication algorithm [10] for which there is no bijection between scalar key values and sequences of curve operations. They randomly switch to an alternative procedure for which multiplications occur for zero bits but not for one bits, and they allow other non-zero digits than 1; $\bar{1}$ for example. This alternative corresponds to a standard recoding

of the input bits to remove long sequences of 1s. On the one hand, the pattern of squares and multiplications is no longer fixed, so that averaging power traces from several exponentiations does not make sense, and, on the other hand, there is ambiguity about which digit value is associated with each multiplication.

This article analyses the set of randomized traces that would be generated by repeated re-use of the same unblinded key k. By aligning corresponding doublings in a number of traces, the possible operation sequences associated with bit pairs and bit triples of the secret key k can be extracted. With only a few traces (10 or so) this provides enough information to determine most bits of k unequivocally, and most of the rest with a high degree of certainty.

Previous work in this area includes [11] and [14]. Oswald [11] takes a similar but *deterministic* algorithm and shows how to determine a space of possible keys from one sequence of curve operations, but not how to combine such results from different sequences. Here the freedom afforded by the randomization minimises the inter-dependence between consecutive operations and so it is unclear whether or not her techniques would lead to an intractable amount of computing. Okeya & Sakurai [14] treat the simple version of the randomized algorithm and succeed in combining results from different multiplications by the same key. They require the key k to be re-used $100+\log_2 k$ times. Here we treat the more complex version of the algorithm, one which is also slightly extended in order to increase security against side channel attacks. The analysis of Okeya & Sakurai is inapplicable in this more general case because it depends on a fixed finite automaton state occuring after processing a zero bit. However, using new methods we find that a) measurements from only $O(10)$ uses of the secret key reveal the key by applying theory which considers pairs of bits at a time, b) software which considers longer sequences of bits can process just two uses to obtain the key in $O(\log k)$ time, and c) for standard key lengths and perfect identification of adds and doubles, a *single* use will disclose the key in a tractable amount of time. In addition, our attack seems less susceptible to error: bits are deduced in parallel so that incorrect deductions of some bits affect at most the neighboring one or two bits. In comparison, the attack of Okeya & Sakurai recovers bits sequentially, making recovery from errors more complex.

Although only one algorithm is studied here, a similar overall approach can be used to break most randomized recoding procedures under the same conditions. The two main properties required are: i) after a given sequence of point operations, the unprocessed part k' of the key can only have one of a small, bounded number of possible values (determined from k by the length of the operation sequence but independent of other choices); and ii) it is possible to identify an associated subset of trace suffixes for which all members correspond to the same value of k'. These also hold for the algorithm proposed by Liardet & Smart [8], which uses a sliding window of random, variable width. They seem to be the key properties required in [20] to demonstrate similar weaknesses in that algorithm also. Two alternatives, MIST [18] and overlapping windows [3], provide a much wider range of values for the unprocessed parts k' of keys. Without property (i) holding, they appear to be safer, particularly when keys are re-used.

Several counter-measures exist for reducing the quantity of data that leaks in this way. For elliptic curve cryptography (ECC), the likelihood of distinguishing between point additions and doublings can be reduced by making each follow the same pattern of field (and other) operations. Several solutions have been proposed [4, 2, 1], but squares and multiplications in the field behave differently [15] and so

there is no reason to believe that such recoding will necessarily hide the distinction between additions and doublings completely.

The attack here depends on the same key being reused on each occasion. One standard counter-measure to frustrate averaging the traces of many exponentiations is to modify the secret exponent each time from e to $e+rg$ where r is a random number, typically 32-bits, and g is the order of the (multiplicative) group in which the exponentiation is performed [6]. Then property (i) above cannot hold. Although this blinding results in a different exponentiation every time, for ECC it adds 20% to the number of point operations for a typical key of 160 bits. It had been hoped that randomized algorithms might avoid this extra cost. This is no longer seems to be the case unless code for adds and doubles can be made to execute indistinguishably.

2 The Oswald-Aigner Exponentiation Algorithm

This section contains a brief outline of the Oswald-Aigner algorithm [10]. It is written in terms of the additive group of points on an elliptic curve E defined over a field F with identity element $\mathcal{O}$. Field elements and rational integers are written in lowercase while points on the curve are written in capitals. The algorithm computes the point $Q = kP$ for a given positive integer k (the secret key) and a given point P on E.

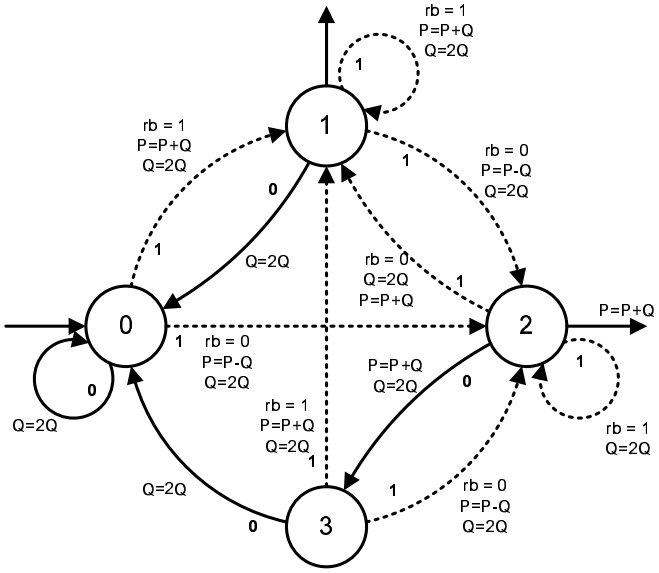

Fig. 1. Finite automaton for an extension of the algorithm. *rb* is a random bit.

The algorithm randomly introduces alternative re-codings to the representation of k. It can be viewed as pre-processing bits of k from right to left into a new digit set $\{-1, 0, +1, +2\}$. Then the resulting scheme for point multiplication can be performed in either direction. The conversion uses a carry bit set initially to 0. When this bit is summed with the current bit of k, the result 0, 1 or 2 can be

re-coded in different ways: 0 always gives a new digit 0 with carry 0; 1 can give either new digit 1 and carry 0 or new digit $\bar{1}$ with carry 1; and 2 gives either new digit 0 and carry 1, or new digit 2 and carry 0.

Fig. 1 illustrates this as a finite automaton for an inconsequential extension of the original algorithm. It has 4 states, numbered 0 to 3 with the carry being 1 if, and only if, the state is 2. For the transition from state 2 to state 1, the normal order of doubling and adding is reversed. This achieves the processing for digit value 2. The extension here allows a new transition from state 0 to state 2; the original algorithm is the special case in which the random bit $rb = 1$ always for state 0. Rather than taking a uniform distribution in every case, we will also allow the random bits to be explicitly biased. However, if the same distribution of random bits is used for each of the states 0, 1 and 3, the automaton can be simplified to consist of just two states, obtained by merging states 0, 1 and 3.

Figure 2 provides equivalent code for the associated right-to-left point multiplication. A left-to-right version is also possible, and can be attacked in the same way.

```
Q ← 𝒪 ;
State ← 0 ;
While k > 0 do
{
   If (k mod 2) = 0 then
   case State of
   {
     0,1,3 : Q ← 2Q ; State ← 0 ;
     2     : P ← P+Q ; Q ← 2Q ; State ← 3 ;
   }
   else
   case State of
   {
     0,1,3 : If rb = 0 then       /* rb is a Random Bit */
                { P ← P-Q ; Q ← 2Q ; State ← 2 }
             else
                { P ← P+Q ; Q ← 2Q ; State ← 1 } ;
     2     : If rb = 0 then       /* rb is a Random Bit */
                { Q ← 2Q ; P ← P+Q ; State ← 1 }
             else
                { Q ← 2Q } ;
   } ;
   k ← k div 2 ;
} ;
If State = 2 then P ← P+Q ;
```

Fig. 2. Oswald & Aigner's randomized signed binary exponentiation (extended).

3 Efficiency Considerations

Although the algorithm appears to be fully described, there are still some details to decide. Specifically, the choice of the random bits rb can be skewed to favour certain transitions. As we shall see, this affects both efficiency and security.

Every bit of the key k is associated with a corresponding doubling operation. This means that the number of doublings is fixed, and time efficiency depends only on the total number of additions and subtractions. We will assume that, for security reasons, point subtractions have been made indistinguishable from point additions as far as is possible. Then efficiency depends entirely on the total number of such operations, which we will henceforth always refer to as additions.

Definition 1. *Let α, β, γ and δ be the chosen probabilities that the random bit rb is 1 when the current state is 0, 1, 2 or 3 respectively.*

Then, for a key k whose bits are selected independently and at random from a uniform distribution, the matrix of transition probabilities between states of the automaton is

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{\alpha}{2} & \frac{\beta}{2} & \frac{1-\gamma}{2} & \frac{\delta}{2} \\ \frac{1-\alpha}{2} & \frac{1-\beta}{2} & \frac{\gamma}{2} & \frac{1-\delta}{2} \\ 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

Lemma 1. *The transition matrix has an eigen-vector $(\frac{1}{2}-\mu, \frac{1}{2}-2\mu, 2\mu, \mu)$ where $\mu = \frac{2-\alpha-\beta}{12-2\alpha-4\beta-4\gamma+2\delta}$ and this contains, as elements, the probabilities associated with each state. Moreover, $0 \le \mu \le \frac{1}{4}$.*

This is an easy exercise for the reader. Taking the dot product of this with the vector $(\frac{1}{2}, \frac{1}{2}, 1-\frac{1}{2}\gamma, \frac{1}{2})$ of average additions associated with each state provides the expected number of additions per bit: $\frac{1}{2}+(1-\gamma)\mu$. Similarly, taking the dot product with the corresponding vector $(1,1,1,1)$ for doublings provides 1 as the expected number of doublings per bit.

In order to minimise the total number of additions and hence the addition chain length, we need to pick $(1-\gamma)\mu = 0$, i.e. $(1-\gamma)(2-\alpha-\beta) = 0$, i.e. either never take the transition from state 2 back to state 1, or never take either of the transitions from states 0 and 1 to state 2. Of course, one might back away from these extremes to retain greater randomness in the chains. In particular, α and/or β should be kept away from 1 so that states 2 and 3 are reachable. In the limit as $\alpha\beta\gamma\delta \to 1$ (which optimises efficiency), on average there is half an addition per bit of k. Thus, a typical addition chain has a little over $\frac{1}{2}\log_2 k$ additions (or subtractions). Even a modest bias towards efficiency, such as taking $\alpha = \beta = \gamma = \delta \ge \frac{3}{4}$, changes this by just 2% or less.

4 The Attack

4.1 Introduction & Initial Hypotheses

The purpose of randomized exponentiation algorithms is to frustrate side channel analysis by an attacker. In particular, they are counter-measures against using knowledge of the exponentiation process to extract the secret key k. Several different levels of leakage are possible, depending on the resources of the attacker. A poor signal-to-noise ratio (SNR) means that many samples have to be taken, and averaging the side channel leakage is one way of improving the SNR. So a critical parameter is whether or not the attacker's equipment is good enough for him to extract sufficient meaningful data from the side channel trace of a single scalar multiplication. If it is, then the standard key blinding described earlier suddenly fails to provide the data hiding protection afforded by averaging away local data dependencies. Improved equipment and laboratory techniques mean that this barrier can now be breached without too much expenditure [12, 13].

The categories of leakage which could be considered include the following: i) individual point operations can be observed on power, EM or other side channel traces; ii) point doublings and point additions can be distinguished from each other; iii) re-use of operands can be observed; and iv) operand addresses can be deduced. Point (i) may hold simply because program instructions and data need to be fetched at the start of each point operation, and these cause different effects on the side channels than field operations. Point (ii) may then hold as a result of different patterns of field operations for point additions than for point doublings. Properties (iii) and (iv) might hold as a result of being able to deduce Hamming weights of data and address words travelling along the bus.

Randomization prevents the obvious averaging of the traces of many point multiplications which was used in initial power analysis attacks on the binary "square-and-multiply" algorithm. There are no longer any corresponding doublings which can be aligned since every point multiplication determines a different sequence of doublings and additions. With matched code for additions and doublings [1, 2, 4, 8], averaging may hide the difference between the two operations because they are no longer separated in time, but in current implementations such averaging will certainly reveal the start and end of the individual point operations which make up the scalar multiplication.

The attack described here requires the SNR to be good enough to extract some useful data from single point multiplications on the curve. Specifically, initially we assume that:

- Adds and doublings can always be identified correctly and distinguished from each other using traces obtained from side channel leakage for a single point multiplication; and
- A number of traces are available corresponding to the same secret key value applied to independent scalar multiplications.

The insistence on "always" in the first hypothesis can be relaxed to provide a more realistic scenario. It is a convenience that allows us to provide a more accurate assessment of the strength of the attack. If doublings and additions can be distinguished successfully with a known probability, then the calculations below can easily be modified appropriately to yield similar results.

4.2 Overview of the Attack & Notation

The outline of the attack is as follows. For simplicity, by the first hypothesis,

– every trace tr is viewed as a word over the alphabet $\{A, D\}$

where A denotes the occurrence of an addition and D the occurrence of a doubling. As usual, the trace is written with time increasing from left to right. However, this is the opposite of the binary representation of the secret key k which is processed *from right to left*, least significant bit first. Consequently, the two face in opposite directions. In particular, if the machine were to cycle round only states 0 and 1 giving the sequence of operations for square-and-multiply exponentiation, then the trace would be essentially the same as the binary, but reversed: every occurrence of 0 would appear as D, and every occurrence of 1 would appear as AD. The binary representation 11001 would then generate the trace $ADDDADAD$. There is one D for every bit, and we index them similarly:

Definition 2. *The* position *of an instance of D in a trace is the number of occurrences of D to its left.*

Thus, the first (leftmost) D of $ADDDADAD$ has position 0 and arises from processing the last (rightmost) bit of 11001, which has index 0.

It is readily verified that the only transition which places D before rather than after an associated occurrence of A is the transition (21). Hence, every occurrence of the substring $DAAD$ in a trace tr corresponds to the processing path traversing the transitions (21) then (12) or (11) in the finite automaton. This substring splits the trace between the As into a prefix and a suffix. There is a corresponding splitting of the binary representation of the secret key k into a least significant part and a most significant part. Since the key and trace are processed in opposite directions with one doubling per transition, i.e. per bit, the least significant part of k has a number of bits equal to the number of Ds in the associated prefix of tr.

Every occurrence of the triplet 111 in k enables the pair AA to occur in some traces (according to the values of the random bits) when processing the second and third least significant bits. Even with a small number of independent traces, almost all triples 111 can be found using these relatively frequent occurrences. It will be shown that similar properties for the distributions of substrings associated with each possible bit pair enable most other bits to be recovered. Since every bit is determined essentially independently of the others, the uncertainties due to noise and errors do not grow unreasonably, and so the attack is feasible in practice.

4.3 Properties of Traces

Lemma 2. *Suppose trace tr is given. If* 11 *occurs at some point in the binary representation of k then the probability of the left-hand* 1 *being represented by transition* (21) *in tr is $\pi = 4\mu(1-\gamma)$.*

Proof. 4μ is the probability of being in state 2 as a result of the right-hand 1 and $1-\gamma$ is the probability of selecting transition (21) next. □

Transition (21) leads to the characteristic pattern $DAAD$ for 111 with which this attack starts. Although there may be other measures to take into account,

as a first approximation we should expect to have to decrease occurrences of this transition in order to maximise security. Over the ranges 0 to 1 for each variable, the probability of the transition is a decreasing function for each of α, β, γ and δ. Hence, to maximise efficiency and security, it would probably be best to choose all of these ≈ 1. This will make $DAAD$ appear only rarely. Of course, choosing $\alpha = 1$ as in the original machine [10] and taking $\beta = 1$ makes states 2 and 3 unreachable, so that all randomization is lost and hence also any security that might accrue from use of the algorithm.

Typically, one might choose all the variables equal to $\frac{3}{4}$, say. This gives the transition (21) a probability of $\pi = \frac{1}{12}$. With this usefully high probability of the associated pattern $DAAD$ in a trace tr, it can reasonably be assumed that almost every occurrence of 111 in k will be recognised from a small set of traces for typical key lengths. For n traces, the probability of not observing $DAAD$ at a given point where 111 occurs is $(1-\pi)^n$. So, to determine 90% of the substrings 111 with these parameter choices, we just need a sample of 27 traces in which the adds and doubles are obtained correctly. If all four variables default to $\frac{1}{2}$, then $\pi = \frac{1}{4}$ and so only 8 traces would enable a similar percentage of the 111s to be established.

In the case of ambiguities or errors in distinguishing additions from doublings, it may be helpful to determine which operations belong to which transition or bit of k. There is one doubling D per bit, but an addition A between consecutive Ds may belong to the transition of either the first bit or the second. Figure 3 lists all possible combinations of transitions for consecutive pairs of bits. Clearly $DAAD$ must split as $DA.AD$ since each transition applies at most one addition. DA only occurs for the transition (21) and so must arise from bit 1. From the available options at state 1 of the automaton, the next bit processed is 1 if, and only if the DA is followed by AD. Otherwise, for the pair 11, the first bit processed is represented by D or AD, and so we know that the D marks the end of processing for that transition. Figure 3 shows all these choices for the bit pair 11. Thus, occurrences of $DAAD$ in some traces enable *all* traces to be split correctly at the point corresponding to the middle of $DAAD$.

To summarize, since DA arises on a single transition only after a preceding 1 has enabled state 2 to be reached,

Lemma 3. *Every occurrence of* $DAAD$ *corresponds to* 111 *in the secret key* k *and enables all traces to be split at the same point. If the middle* 1 *has index* i *then the subtraces corresponding to the bits from index* 0 *to* i *are given by all characters up to and including the* D *at position* i*, plus the next* A *when two* A*s occur (but no additional* A *otherwise).*

To recover more of k, we look next at other pairs of bits, and see how many As can and do appear in traces between the two Ds corresponding to the bits. As just noted, there can be up to 2 intervening As between consecutive Ds.

Lemma 4. i) *For a given trace, if the* D*s in positions* i *and* $i{+}1$ *are not separated by any* A*s, then the bit pair* $k_{i+1}k_i$ *is* 00 *with probability* $(2{-}2\mu(1{-}\gamma))^{-1}$*, which exceeds* $\frac{1}{2}$*. If the* D*s are separated by one or more* A*s in any trace, then the bit pair is certainly not* 00*.*

ii) *For a given trace, if the Ds in positions i and $i+1$ are separated by one A, then the bit pair $k_{i+1}k_i$ is* 10 *with probability* $\frac{1}{2}$. *If the Ds are separated by no As or two As in any trace, then the bit pair is certainly not* 10.
iii) *For a given trace, if the Ds in positions i and $i+1$ are separated by two As, then the bit pair $k_{i+1}k_i$ is certainly* 11. *The probability of two As when the bit pair is* 11 *is* $2\mu(1-\gamma)$, *assuming bit k_{i-1} is unknown.*
iv) *For a set of n traces, suppose the Ds in positions i and $i+1$ are separated by no As in some cases, by one A in some cases, and by two As in no cases. Then the bit pair $k_{i+1}k_i$ is* 01 *with probability* $(1+(1-2\mu(1-\gamma))^n)^{-1}$.

Proof. i) First, by inspection of the finite automaton, we see that the only possible operation sequences for 00 are ADD and DD. So the Ds are always adjacent. The intervention of an A will prove that the bit pair is not 00.

Bit Pair	*Operation Patterns*	*State Sequences*	*Probabilities, given the bit pair*
00	$D.D$	000, 100, 300	$1-2\mu$
	$AD.D$	230	2μ
10	$D.AD$	001, 002	$\frac{1}{2}-\mu$
	$D.AD$	101, 102	$\frac{1}{2}-2\mu$
	$AD.AD$	231, 232	2μ
	$D.AD$	301, 302	μ
01	$AD.D$, $AD.AD$	010, 023	$(\frac{1}{2}-\mu)\alpha$, $(\frac{1}{2}-\mu)(1-\alpha)$
	$AD.D$, $AD.AD$	110, 123	$(\frac{1}{2}-2\mu)\beta$, $(\frac{1}{2}-2\mu)(1-\beta)$
	$DA.D$, $D.AD$	210, 223	$2\mu(1-\gamma)$, $2\mu\gamma$
	$AD.D$, $AD.AD$	310, 323	$\mu\delta$, $\mu(1-\delta)$
11	$AD.AD$, $AD.AD$	011, 012	$(\frac{1}{2}-\mu)\alpha\beta$, $(\frac{1}{2}-\mu)\alpha(1-\beta)$
	$AD.DA$, $AD.D$	021, 022	$(\frac{1}{2}-\mu)(1-\alpha)(1-\gamma)$, $(\frac{1}{2}-\mu)(1-\alpha)\gamma$
	$AD.AD$, $AD.AD$	111, 112	$(\frac{1}{2}-2\mu)\beta^2$, $(\frac{1}{2}-2\mu)\beta(1-\beta)$
	$AD.DA$, $AD.D$	121, 122	$(\frac{1}{2}-2\mu)(1-\beta)(1-\gamma)$, $(\frac{1}{2}-2\mu)(1-\beta)\gamma$
	$DA.AD$, $DA.AD$	211, 212	$2\mu(1-\gamma)\beta$, $2\mu(1-\gamma)(1-\beta)$
	$D.DA$, $D.D$	221, 222	$2\mu\gamma(1-\gamma)$, $2\mu\gamma^2$
	$AD.AD$, $AD.AD$	311, 312	$\mu\delta\beta$, $\mu\delta(1-\beta)$
	$AD.DA$, $AD.D$	321, 322	$\mu(1-\delta)(1-\gamma)$, $\mu(1-\delta)\gamma$

Fig. 3. All possible operation sequences for all bit pairs, and probabilities given the bit pair occurs. (*The bit pairs are processed right to left.*)

Figure 3 shows the operation sequences which can occur for each bit pair. It includes the probability of each, assuming that the bit pair occurs and that the initial states have the probabilities determined by Lemma 1. Knowledge of neighbouring bits would require these values to be modified.

Suppose there is no intervening A between the two specified Ds. If the bit pair is 00 then the probability of this is $\pi_{00} = 1$; if the bit pair is 10 then the probability is $\pi_{10} = 0$; if the bit pair is 01 then the probability is $\pi_{01} = (\frac{1}{2}-\mu)\alpha+(\frac{1}{2}-2\mu)\beta+\mu\delta$; and if the bit pair is 11 then the probability is $\pi_{11} = (\frac{1}{2}-\mu)(1-\alpha)+(\frac{1}{2}-2\mu)(1-\beta)+2\mu\gamma+\mu(1-\delta)$. Thus, the correct deduction of 00 is made with probability

$$\pi_{00}/(\pi_{00}+\pi_{10}+\pi_{01}+\pi_{11}) = 1/(2-2\mu(1-\gamma)).$$

ii) From the table or the automaton, we can see that the bit pair 10 must always include the operation A once between the two occurrences of D, but this is not the case for any other bit pair. Thus the absence of an A, or the presence of two As, guarantees the bit pair is not 01. However, suppose there is exactly one A between the specified Ds. If the bit pair is 00 then the probability of this is $\pi'_{00} = 1-\pi_{00} = 0$; if the bit pair is 10 then the probability is $\pi'_{10} = 1-\pi_{10} = 1$; if the bit pair is 01 then the probability is $\pi'_{01} = 1-\pi_{01}$; and if the bit pair is 11 then the probability is $\pi'_{11} = 1-\pi_{11}-2\mu(1-\gamma)$. Thus, the correct deduction of 10 is made with probability

$$\pi'_{10}/(\pi'_{00}+\pi'_{10}+\pi'_{01}+\pi'_{11}) = \tfrac{1}{2}.$$

iii) This part is clear from the table in Figure 3.
iv) Finally, by parts (i) and (ii), a bit pair which includes both the possibilities of no As and of one A between the specified Ds cannot be 00 or 10; it must be 01 or 11. The probability of not having two As in any trace when the digit pair is 01 is 1, of course. The probability of not having two As in any of the n traces when the digit pair is 11 is $\pi_n = (1-2\mu(1-\gamma))^n$. Hence the probability of the pair being 01 rather than 11 is $1/(1+\pi_n)$. □

We must be careful in the application of this lemma. Each part assumes no knowledge of bit k_{i-1}. Knowing it changes the probabilities. In most cases, the differences are small enough to be considered negligible; for accurate figures the table can be used to select just the cases starting in states 0 or 3 when the preceding processed bit is 0, and the cases starting in states 1 or 2 when that bit is 1. The only case where a qualitative difference occurs is for 11 when AA only occurs if $k_{i-1} = 1$. In the case of $k_{i-1} = 0$ this means we cannot distinguish 01 from 11 so easily. However, no use has yet been made of the *distribution* of characters adjacent to, or between, the two Ds of interest: for example, using the relative probabilities, one could distinguish correctly between the pairs 01 and 11 with a better than evens chance on the basis of the proportion of As which occur between the Ds. This presupposes knowledge of the values of the parameters α, β, γ and δ, but, if necessary, these can be deduced with sufficient accuracy from the relative frequencies of various patterns in the traces. We will not go into detail here since the aim is to establish the feasibility of the attack, rather than the minimal number of traces required for a given chance of success. This last is, in any case, dependent on the counter-measures employed by the crypto-system and the quality of the techniques and monitoring equipment used by the attacker.

4.4 Reconstructing the Key

From parts (i) and (ii) of Lemma 4, we can establish every occurrence of the bit pairs 00 and 10 with any desired degree of confidence if a sufficiency of independently generated traces for the same key is available.

The probability that a 1 bit belongs to one of these pairs 00 or 10 is $\frac{1}{2}$ because a 1 has 50% chance of being followed by a 0 to give 10. Also, the probability that a 0 bit belongs to one of the pairs 00 or 10 is 1 because it must be preceded by a 0 or a 1 to produce 00 or 10 respectively. Hence, with enough traces, $\frac{3}{4}$ of the bits of k will be established easily and correctly using Lemma 4, (i) or (ii). Half of all bits belong to exactly one of these pairs, and a quarter of all bits belong to two such pairs (namely, in the latter case, the central bit of triplets of the form $*00$). On average, half of all bit pair positions are used to determine these bits of k using the first two parts of the lemma.

For isolated bit pairs 00 and 10, i.e. those which do not overlap with another bit pair of the same type, the lemma shows that n traces would lead to less than one error per 2^n deductions of the pair. However, for those bits belonging to a longer sequence of overlapping such bit pairs, the probability of a correct deduction is much higher. Such sequences are maximal substrings of the form 10^i if no errors have been made. In an infinitely long sequence of independent random bits, $\frac{1}{4}$ of 0s would belong to a sequence 101, so about $\frac{1}{4}$ of the bits of k are determined as isolated pairs. The other $\frac{3}{4}$ of 0s and $\frac{1}{4}$ of 1s belong to longer sequences of overlapping pairs. This increases the probability of correct deductions for half the bits of k because of the need for consistency where pairs overlap. We might expect fewer than 1 error in 2^{2n} for these bits, i.e. half of all bits of k are determined with almost complete certainty. This could be established by extending Figure 3 and Lemma 4 from pairs to triples of bits, but this level of detail is unnecessary to prove the feasibility of the attack.

Overall, for a standard 192-bit key k, an average of 96 bit pair positions would have trace operations which determine the values of 144 bits. So, with only 10 traces, we can expect every such pair to be determined correctly because substantially less than one error per 2^{10} pairs should be made. Thus, all determined bits will be correct for at least 9 out of every 10 keys which are attacked.

A quarter of the bits now remain to be determined. The number of traces required to establish these with a given probability depends much more on the choice of the parameters α to δ than was the case for the previous bits. To determine them, we consider sequences of 1s rather than 0s. $i(\frac{1}{2})^{i+1}$ of 1 bits belong to maximal sequences of exactly i consecutive 1 bits, i.e. to subsequences 01^i0. Hence $\frac{1}{2}$ of all 1 bits belong to sequences of 3 or more 1 bits. Lemma 2 enables us to determine these bits with a known probability for a given number of traces. Default values for the parameters α to δ would mean that around 90% of triples 111 would be detected using just 8 traces, whatever the key length. For convenience, assume the choice of parameters and number of traces allow this percentage to be determined. Since some of the triples will overlap, well over 90% of the 1s in such triples should be identified. In fact, $\frac{1}{4}$ of all 1s lie at the ends of sequences of 3 or more 1 bits, $\frac{1}{4}$ of all 1s lie within such sequences, and $\frac{1}{16}$ of all 1s are the central bit of a subsequence 01110. Hence $\frac{3}{16}$ of all 1s are internal 1s which belong to at least two triples 111, and so at least 99% of these will be determined correctly. Another $\frac{5}{16}$ of all 1s lie in a single triple 111, and, by assumption, 90% of them are determined as a result of this. None of the initial or internal 1s in a subsequence 01^i0 ($i \geq 3$) were considered in the previous paragraphs since they are not contained in either of the pairs 00 or 10. So these 1 bits add almost another $\frac{1}{16}+\frac{1}{8}$ to the fraction of all bits determined so far. On average 10% of these 1s will be undetermined because $DAAD$ does not occur in any trace. So about one of these bits will fail to be spotted in a 192-bit key.

This leaves unconsidered only those $\frac{1}{16}$th of all bits which are the initial 1s of sequences 0110. In a sequence of the form $0*10$, the component 0s are determined as above, as is the constituent pair 10. If the remaining bit $*$ were a 0 then the subsequence 00 would occur, it would be identified, and so $*$ determined as a 0. Since $*$ is not so determined, $*$ must be 1. So all bits are now determined except for some of those which belong to the triples 111 that failed to exhibit a substring $DAAD$ in any trace. Thus,

Theorem 1. *Suppose elliptic curve adds and doubles can be distinguished accurately on a side channel. If the Oswald-Aigner exponentiation algorithm is used with the same unblinded* 192*-bit ECC key* k *for* 10 *point multiplications then all of a known set of about* $\frac{3}{4}$ *of the bits can be determined correctly with probability exceeding* $\frac{9}{10}$*. With a uniform distribution of random bits for choosing the re-coding, all but about one or two of the remaining bits can be determined with at least similar confidence. The undetermined bits are in known positions.*

This theorem says that a standard 192-bit key can usually be broken on a first attempt using a dozen traces with essentially no computational effort beyond extracting the add and double patterns from each trace. By considering all possibilities, the few undetermined bits lead to a very small set of possible keys, of which the correct one can surely be established by decrypting some ciphertext. Identical working shows that a similar theorem holds for keys of any length. In all cases the number of traces needed to achieve a very high, specified degree of confidence in the determined bits is $O(\log \log k)$.

4.5 Secure Parameter Choices?

Varying the parameters may increase the difficulty of determining the quarter of bits not covered by (i) and (ii) of Lemma 4. The probability of the sequence $DAAD$ has to be decreased for this. This requires $\mu \to 0$ or $\gamma \to 1$.

For the first of these, the probability of states 2 and 3 falls towards 0. Then the traces closely match the pattern of operations for square-and-multiply. So, roughly speaking, bits are determined according to the whether many or few traces have A before the corresponding D. With Lemma 4 determining most occurrences of 00 and 10, the only outstanding problem is to distinguish between 01 and 11. This is easily resolved by considering the average number of As between the relevant pair of Ds: μ is very small, so α and β are close to 1 and therefore 01 has no As almost always whereas 11 has one A with the same probability, i.e. almost always, if no AAs appear. Thus $\mu \approx 0$ is not a secure option.

Bit Pair	*Operation Patterns*	*Probabilities, given the bit pair*
00	$D.D$	$1-2\mu$
	$AD.D$	2μ
10	$D.AD$	$1-2\mu$
	$AD.AD$	2μ
01	$AD.D$	$(\frac{1}{2}-\mu)\alpha+(\frac{1}{2}-2\mu)\beta+\mu\delta$
	$AD.AD$	$(\frac{1}{2}-\mu)(1-\alpha)+(\frac{1}{2}-2\mu)(1-\beta)+\mu(1-\delta)$
	$D.AD$	2μ
11	$AD.AD$	$(\frac{1}{2}-\mu)\alpha+(\frac{1}{2}-2\mu)\beta+\mu\delta$
	$AD.D$	$(\frac{1}{2}-\mu)(1-\alpha)+(\frac{1}{2}-2\mu)(1-\beta)+\mu(1-\delta)$
	$D.D$	2μ

Fig. 4. Operation sequences and probabilities for bit pairs when $\gamma = 1$.

Alternatively, we must take $\gamma \approx 1$. Then the table of Fig. 3 reduces to that of Fig. 4. Unless the probablities are carefully chosen, each bit pair will be distinguishable from the others by the occurrences of fixed or variable numbers of As between the corresponding Ds: 00 contains no As, 10 contains one A, and the other two have instances of both no A and one A between the Ds. As in the previous case, 01 and 11 will be distinguishable because they will have different, complementary probabilities for the occurrences of no As. Hence, in order to retain some ambiguity, it is necessary to force 01 and 11 to have fixed numbers of As between their Ds (so they are indistinguishable from 00 or 10) or for them to have variable numbers which occur with equal probability $\frac{1}{2}$ (so they are indistinguishable from each other).

By assumption, μ is not close to 0. Hence the last case in Fig. 4 for each of 01 and 11 is non-negligible. Thus 01 and 11 could only demonstrate a fixed number of As over the set of traces if their first cases were to occur with negligible probability. Since $0 < \mu \leq \frac{1}{4}$, this would require $\alpha \approx \delta \approx 0$, which makes $\mu \approx \frac{1}{4}$. Then 00 and 11 may be confused, as may 01 and 10. However, the traces can now be used to determine the parity of one bit from that of its neighbour – either equal parity or opposite parity. Since the leading bit is 1, the parity of every bit is determined, and so the secret key k can be recovered. If there is an error, it must arise from confusing As with Ds and vice versa. However, such errors can be determined simply by taking several traces: if the number of As varies between two Ds, there must be an error. In fact, as long as A and D can be obtained correctly with greater probability than by guessing, it just requires enough traces to determine with any required certitude whether the two bits of the pair are equal or not. Hence, neither is this a secure solution.

The last possible way for perhaps keeping some ambiguity is by picking the probabilities so that 01 and 11 are confused. This requires the first case of each in Fig. 4 to have the same probability as that of the combination of the second and third cases, namely $\frac{1}{2}$ each. Choosing $\gamma \approx 1$ effectively removes the transition from state 2 to state 1. Thus DA will not occur for any bit. This means that every A belongs to the following D. Thus, every trace can be correctly parsed into substrings D or AD which correspond to whole transitions. In particular, this means that *all* characters of the operation patterns listed in Fig. 4 are known, not just those between the Ds. So 01 and 11 can be distinguished from each other using the occurrences of $D.AD$ and $D.D$ unless μ is close to 0. However, $\mu \approx 0$ has already been rejected as a possibility. That choice would make the algorithm revert to square-and-multiply: 01 and 11 could be distinguished by the different probabilities of $.D$ and $.AD$ in Fig. 4, corresponding to the first bit being 0 or 1 respectively. In fact, $\mu = 0$ forces $\alpha = \beta = 1$, so that the $.AD$ has probability 0 for 01 and probability 1 for 11.

It is now clear that any choice of parameters produces sufficient differences between the sub-traces associated with pairs of bits for every pair to be identified correctly when enough traces are provided. Whenever the parameters are chosen to remove one distinguishing feature, another feature appears, enabling them to be distinguished again. In this way all the bit pairs are determined with a known certainty. This certainty is amplified by the fact that each bit pair overlaps two other bit pairs and the determinations must be consistent with each other.

Lack of space precludes full details, but the outline feasibility proof for these exceptional cases with reasonable key lengths is as follows. As in the proof of Theorem 1, each trace determines each bit (or bit pair) correctly with a given probability greater than some non-zero ϵ. As above, by selecting suitable criteria which

depend on the choice of parameters, ϵ can be made noticably larger than 0 in all cases. Indeed, from Lemma 4, it is 1 in some cases and above $\frac{1}{2}$ for many others. With n independently generated traces, each bit will be known with probability at least $1-\bar{\epsilon}^n$. To ensure an average of at most $c = O(1)$ bits are incorrect, n must be picked large enough to satisfy $1-\bar{\epsilon}^n > 1-\frac{c}{\log k}$, and this requires n to be $O(\log\log k)$. However, we can reasonably assume that it is computationally feasible to check all possibilities for every set of $c \leq 3.5$ bits which might be in error: it requires checking $O((\frac{1}{4}\log k)^{2^c})$ keys because essentially three quarters of the bits are already known. Thus,

Theorem 2. *No choice of algorithm parameters is secure for a reasonable key length under the above attack if* $O(\log\log k)$ *traces are available from point multiplications using the same unblinded key.*

4.6 Counter-Measures

In the absence of a secure set of parameter choices, further counter-measures are required. The most obvious counter-measure is to restore key blinding. A small number of blinding bits might still result in the attacker's desired 10 or so traces for the same key eventually becoming available. These might be identified easily within a much larger set of traces by the large number of shared characters between their traces. So, to avoid such duplication, the size of the random number used in blinding cannot reasonably be much less than the maximum lifespan of the key in terms of the maximum number of point multiplications it is used for. Thus 16 or more bits are needed, and this will add 10% or more to the cost of point multiplication.

Identical formulae for additions and doublings are increasingly efficient and applicable to wider classes of elliptic curves. We single out those of Brier and Joye [1] in particular. However, the potential to deduce Hamming weights of bus data from side channel leakage may still enable attackers to discriminate adds and doubles from address or data loading cycles.

Another favoured counter-measure is the add-and-always-double approach which is applied to m-ary exponentiation. Then each occurrence of DD has an add inserted to yield the pattern DAD, but the add output is discarded without having been used. This should also be done for the Oswald-Aigner algorithm. In addition, an extra double needs to be performed to convert each $DAAD$ into $DADAD$, and the double's output is likewise ignored.

Alternatives randomized algorithms exist. Standard m-ary exponentiation [5] is not subject to this type of attack, but re-use of operands might be detected [16], making that method unsuitable even with key blinding in place. The MIST algorithm [17–19] and an overlapping windows method [3] currently seem to be the most robust choices under these types of attack.

5 Overview

The introduction presented an overview of two algorithm properties which appear to be necessary for the above type of attack. Here, and in the attacks of [14] and [20], the randomized algorithms always process the key through a subset of a small, fixed set of values: after each iteration of the key processing algorithm, the scalar which still needs processing is simply a prefix of the binary representation of the key

plus a small carry. Thus, at any given point in the processing, the corresponding sub-traces must all correspond to the same key bits plus the carry. Consequently, unless perhaps the algorithm is extremely well-designed, it can be assumed that the different frequencies of the various trace patterns will reveal the correct key bit pattern with a confidence which increases with the number of traces. No cases require more than $O(10)$ traces to recover the key. Thus, such randomized algorithms are probably best used only for ephemeral keys unless there is confidence in the efficacy of other counter-measures.

6 One Trace

It is interesting to speculate on how much data leaks from a single point multiplication since this may now be the main context for the algorithm. Oswald [10] noted that for some *deterministic* re-coding algorithms in which several non-zero digits generate indistinguishable As, the operation patterns resulting from numbers of up to 12 bits could only represent at most 3 keys. By breaking a standard ECC key into 12 bit sections, this means very few keys actually generate an observed patterns of operations. Moreover, these can be ordered according to their liklihood of occurrence, and this considerably reduces the average search time for the correct key. Hence the key can be recovered quite easily.

She also writes that the same attack is possible on randomized algorithms with weaker results, but provides no detail. Randomized algorithms have much weaker inter-dependencies between adjacent operation patterns. This should substantially increase the number of keys which match a specific pattern of point operations. The key Lemma 4 above does not provide much certainty for any bits unless a number of traces are available; only the infrequent instances of AA allow definite determination of any bits. Of course, an analysis of subsequences of more than two bits is possible, as in [14], but, besides better probabilities, this gives no further insight. Instead, software was written to enumerate all the keys which could represent a given string. On average, for the extended version of the algorithm, the trend up to 16-bit keys indicates clearly that a little over $O(\sqrt[4]{k})$ keys will match a given pattern – under 20 match a given 16-bit pattern. This would appear to ensure the strength of the algorithm when a key is used just once but only if the key has at least 2^8 bits or there is considerable ambiguity in the side channel about whether the operations are adds or doubles. The original algorithm has fewer random choices, and so has even fewer keys matching a given pattern.

With only two traces available for the same key, an attacker can use occurrences of D to partition the traces into small corresponding sections which represent the same key bits. From the two sets of matching key values for each section, he selects pairs which differ only by the possible carry values at each end of the section. Then, choosing the cases where the carry values match between sections, he concatenates the possible bit strings for each section to recover the whole key. This is most likely to produce a very small number of possible keys, and certainly a tractable number for reasonable key lengths.

7 Conclusion

One of several, similar, randomized exponentiation algorithms has been investigated to assess its strength against a side channel attack which can differentiate between point additions and point doublings. Straightforward theory shows that at most $O(10)$ uses of the same unblinded key will enable the secret key to be recovered easily in a computationally feasible time. No choice of parameters improves security enough to alter this conclusion. Using longer bit sequences than the theory, it is also clear that software can search successfully for keys when just 2 side channel traces, and perhaps only one, are available. However, this number may need increasing if adds and doubles might be confused or standards for key lengths are increased.

The main property which is common to algorithms which can be attacked in this way seems to be that the next subsequence of operations at a given point in the processing of the key must be chosen from a small, bounded set of possibilities which is derived from the key and the position, but is independent of previous choices. Hence, our overall conclusion is that such algorithms should be avoided for repeated use of the same unblinded key if adds and doubles can be differentiated with any degree of certainty. Furthermore, for typical ECC key lengths, a single use may be sufficient to disclose the key when adds and doubles are accurately distinguishable.

References

1. E. Brier & M. Joye, *Weierstraß Elliptic Curves and Side-Channel Attacks*, Public Key Cryptography, P. Paillier & D. Naccache (editors), Lecture Notes in Computer Science, **2274**, Springer-Verlag, 2002, 335–345.
2. C. Gebotys & R. Gebotys, *Secure Elliptic Curve Implementations: An analysis of resistance to power-attacks in a DSP processor core*, Cryptographic Hardware and Embedded Systems (CHES 2002), Lecture Notes in Computer Science, **2523**, Springer-Verlag, 2003, *to appear*.
3. K. Itoh, J. Yajima, M. Takenaka, & N. Torii, *DPA Countermeasures by improving the window method*, Cryptographic Hardware and Embedded Systems (CHES 2002), Lecture Notes in Computer Science, **2523**, Springer-Verlag, 2003, *to appear*.
4. M. Joye & J.-J. Quisquater, *Hessian Elliptic Curves and Side Channel Attacks* Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 402–410.
5. D. E. Knuth, *The Art of Computer Programming*, vol. **2**, "Seminumerical Algorithms", 2nd Edition, Addison-Wesley, 1981, 441–466.
6. 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.
7. 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.
8. P.-Y. Liardet & N. P. Smart, *Preventing SPA/DPA in ECC Systems using the Jacobi Form* Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. Koç, D. Naccache & C. Paar (editors), Lecture Notes in Computer Science, **2162**, Springer-Verlag, 2001, 391–401.
9. 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.

10. 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.
11. E. Oswald, *Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems*, Cryptographic Hardware and Embedded Systems (CHES 2002), Lecture Notes in Computer Science, **2523**, Springer-Verlag, 2003, *to appear*.
12. 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.
13. J.-J. Quisquater & D. Samyde, *Eddy current for Magnetic Analysis with Active Sensor*, Proc. Smart Card Programming and Security (e-Smart 2002), Nice, September 2002, 183–194.
14. K. Okeya & K. Sakurai, *On Insecurity of the Side Channel Attack Countermeasure using Addition-Subtraction Chains under Distinguishability between Addition and Doubling*, Information Security and Privacy (ACISP 2002), L. Batten & J. Seberry (editors), Lecture Notes in Computer Science, **2384**, Springer-Verlag, 2002, 420–435.
15. 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.
16. 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.
17. C. D. Walter, *Improvements in, and relating to, Cryptographic Methods and Apparatus*, UK Patent Application 0126317.7, Comodo Research Laboratory, 2001.
18. C. D. Walter, *MIST: An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis*, Proceedings of CT-RSA 2002, Lecture Notes in Computer Science, **2271**, Springer-Verlag, 2002, 53–66.
19. C. D. Walter, *Some Security Aspects of the MIST Randomized Exponentiation Algorithm*, Cryptographic Hardware and Embedded Systems (CHES 2002), Lecture Notes in Computer Science, **2523**, Springer-Verlag, 2003, *to appear*.
20. C. D. Walter, *Breaking the Liardet-Smart Randomized Exponentiation Algorithm*, Proc. Cardis '02, San José, November 2002, USENIX Association, Berkeley, 2002, 59–68.