# Issues of Security with 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. Oswald and Aigner describe such an algorithm for point multiplication in elliptic curve cryptography (ECC). Assuming an attacker can identify and distinguish additions and doublings 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. Thus blinding should still be used or great care taken to minimise the differences between point additions and doublings.

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

## 1 Introduction

Side channel attacks [6, 7] on embedded cryptographic systems show 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 crypto-systems which use the computationally expensive function of exponentiation, such as RSA, ECC and Diffie-Hellman. Early attacks required averaging over a number of exponentiations [9] to extract meaningful data, but improved techniques mean that single exponentiations using traditional algorithms may be insecure. In particular, it should be assumed that the pattern of squares and multiplies can be extracted fairly accurately from side channel leakage, perhaps by using Hamming weights to identify operand re-use. Where the standard binary "square-and-multiply" algorithm is used, this pattern reveals the secret exponent immediately.

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 a different procedure for which multiplications appear to occur instead for zero bits but not for one bits. This alternative corresponds to a standard recoding of the input bits to remove long sequences of 1s and introduces other non-zero digits such as $\bar{1}$. 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 (ten or so) this provides enough information to determine half the bits of $k$ unequivocally, and the rest with a very high degree of certainty.

Previous work in this area includes [11] and [14]. In [11] Oswald 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 randomization minimises the inter-dependence between consecutive operations and so it is unclear whether or not her techniques 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 in an extended form which might increase security. The analysis of Okeya & Sakurai is inapplicable here because it depends on a fixed finite automaton state occurring 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: key bits are deduced independently so that any incorrect deductions affect at most the neighbouring 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 [16] to demonstrate similar weaknesses in that algorithm.

Several counter-measures exist against this type of attack. As well as standard blinding by adding a random multiple of the group order to the exponent, different algorithms can be employed, such as [3, 5]. Moreover, formulae for point additions and doublings can be made indistinguishable [1, 2, 4, 8].

## 2 The Oswald-Aigner Exponentiation Algorithm

This section contains a brief outline of the Oswald-Aigner algorithm [10] in terms of the additive group of points on an elliptic curve $E$. Rational integers are written in lowercase while points on the curve are written in capitals and Greek characters denote probabilities. The algorithm computes the point $P = kQ$ for a given positive integer $k$ (the secret key) and a given point $Q$ on $E$.

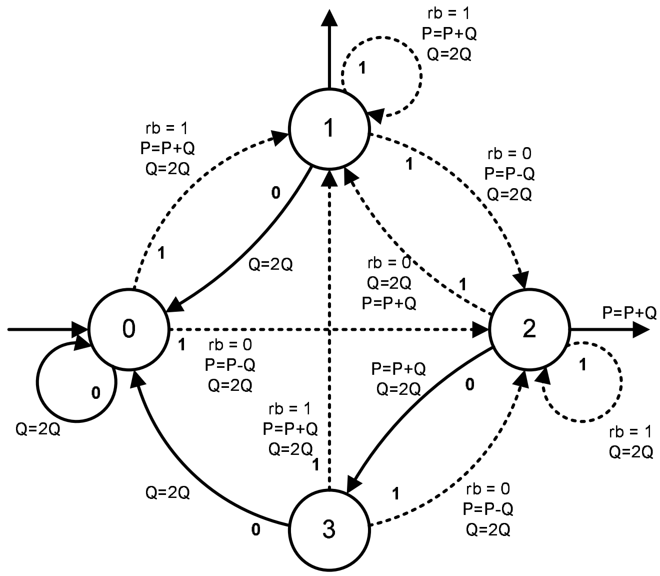


**Fig. 1.** Finite automaton for an extension of the algorithm. $rb$ is a random bit.

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 a slight extension of the original right-to-left 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. The extension also allows the random bits to be biased for each state. However, if the same distribution of random bits is used for each of the states 0, 1 and 3, the automaton simplifies to 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.

```
P ← 𝒪 ;                    /* 𝒪 is the zero of the elliptic curve */
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

**Definition 1.** *Let $\alpha$, $\beta$, $\gamma$ and $\delta$ be the probabilities that the random bit $rb$ is chosen to be* 1 *when the current state is* 0, 1, 2 *or* 3 *respectively.*

These probabilities can be chosen to improve efficiency or, as we shall see, security. 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 then

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

**Lemma 1.** *The transition matrix has an eigen-vector* $(\frac{1}{2}-\mu, \frac{1}{2}-2\mu, 2\mu, \mu)$ *where* $\mu = \frac{2-\alpha-\beta}{12-2\alpha-4\beta-4\gamma+2\delta}$. *Its elements are 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 eigen-vector 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: $\frac{1}{2}+(1-\gamma)\mu$. The number of doublings is constant at one per bit. So, to minimise the total time we require $(1-\gamma)\mu = 0$, i.e. $(1-\gamma)(2-\alpha-\beta) = 0$, i.e. disallow either the transition from state 2 back to state 1, or both transitions to state 2 from states 0 and 1. Avoiding these extremes provides greater randomness. In particular, $\alpha$ and/or $\beta$ 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 Initial Hypotheses, Notation & Overview of the Attack

The attack here assumes sufficiently good monitoring equipment and a sufficiently weak implementation. Specifically it is assumed that:

- Adds and doublings can always be identified and distinguished correctly in side channel leakage from a single point multiplication; and
- Side channel traces are available for a number of different uses of the same, unblinded key value.

For ease in calculating probabilities, we assume adds and doublings can *always* be distinguished. Similar results hold if this is only usually the case. By the first hypothesis,

- every side-channel trace $tr$ can be viewed as a word over the alphabet $\{A, D\}$

where $A$ denotes the occurrence of an addition and $D$ that of a doubling. Here, as expected, 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*, so that the re-coding can be done on the fly (Fig. 2). For example, 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 $AD$. So the binary representation 11001 would generate the trace $ADDDADAD$. There is one $D$ for every bit, and we index them to correspond:

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

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

The attack consists of a systematic treatment of observations like the following. 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 traversing transition (21) then (12) or (11) in the finite automaton. This must correspond to processing a bit 1 to reach state 2, and then two further 1 bits. The trace can be split between the two adjacent $A$s into a prefix and a suffix. There is a corresponding split in the binary representation of the secret key $k$ such that the suffix of $k$ has a number of bits equal to the number of $D$s in the prefix of $tr$. This enables the position of the substring 111 to be determined in $k$. Moreover, by the next lemma, most occurrences of 111 can be located in this way if enough traces are available: $DAAD$ appears exactly when the middle 1 is represented by the transition (21).

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

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

### 4.2 Properties of the Traces

Figure 3 lists the transitions and operation sequences which can occur for each bit pair, including the probability of each. It assumes that initial states have the probabilities determined by Lemma 1, and that neighbouring bits are unknown. The figure enables one to see which bit pairs can arise from given patterns in a trace, and to calculate their probabilities:

**Lemma 3.** *Let $k_i$ denote the bit of $k$ with index $i$, and $\mu$ be as in Lemma 1. Then,*

i) *For a given trace, if the $Ds$ in positions $i$ and $i+1$ are not separated by any $As$, then the bit pair $k_{i+1}k_i$ is* 00 *with probability* $(2-2\mu(1-\gamma))^{-1}$, *which is at least* $\frac{1}{2}$. *If the $Ds$ are separated by one or more $As$ 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* $(1{+}(1{-}2\mu(1{-}\gamma))^n)^{-1}$.

| *Bit Pair* | *Operation Patterns* | *State Sequences* | *Probabilities, given the bit pair* |
|---|---|---|---|
| 00 | $D.D$ | 000, 100, 300 | $1{-}2\mu$ |
| | $AD.D$ | 230 | $2\mu$ |
| 10 | $D.AD$ | 001, 002 | $\frac{1}{2}{-}\mu$ |
| | $D.AD$ | 101, 102 | $\frac{1}{2}{-}2\mu$ |
| | $AD.AD$ | 231, 232 | $2\mu$ |
| | $D.AD$ | 301, 302 | $\mu$ |
| 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 their probabilities given the bit pair occurs. (*Bit pairs are processed right to left and operations left to right.*)

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

Suppose there is no intervening $A$ between the two specified $D$s. Using Figure 3, 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

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

ii) Similarly, from Figure 3 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 $A$s, guarantees the bit pair is not 10. However, suppose there is exactly one $A$ between the specified $D$s. By Figure 3, 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 immediate from Figure 3.

iv) Finally, by parts (i) and (ii), a bit pair which includes both the possibilities of no $A$s and of one $A$ between the specified $D$s cannot be 00 or 10; it must be 01 or 11. The probability of not having two $A$s in any trace when the digit pair is 01 is 1, of course. By Fig. 3 the probability of not having two $A$s 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 a little careful in the application of this lemma. Firstly, 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. This is a typical problem to solve when reconstructing the whole key.

Secondly, deductions from different traces are not independent. For example, suppose all of $n$ traces have one $A$ between the $D$s in positions $i$ and $i+1$. From (ii) of the lemma it is tempting to deduce that the bit pair is 10 with probability $1-(\frac{1}{2})^n$. However, the probability of this may still only be $\frac{1}{2}$. In particular, this happens when the parameters $\alpha = \beta = \delta = 0$ are selected. Then the bit pairs 10 and 01 would always have exactly one $A$ between the $D$s, and bit pairs 00 and 11 would never have any $A$s. So 01 and 10 would be equally likely with probability $\frac{1}{2}$ if exactly one $A$ always occurred. The independent decisions which *can* be combined are those based on the independent choices of random bits, as in (iv).

### 4.3 Reconstructing the Key

For this section we assume the default values which give the original algorithm, namely $\alpha = 1$ and $\beta = \gamma = \delta = \frac{1}{2}$. This means $\mu = \frac{1}{14}$. Later we consider alternatives which might improve security. Then Figure 3 immediately yields:

**Lemma 4.** *For the above default values of the parameters,*
*i) the bit pair* 01 *has no intervening* $A$ *between the associated* $D$*s of a trace with probability* $\frac{9}{14}$ *and one intervening* $A$ *with probability* $\frac{5}{14}$*;*

*ii) the bit pair* 11 *has no intervening $A$ between the associated $D$s with probability* $\frac{2}{7}$*, one intervening $A$ with probability* $\frac{9}{14}$*, and two $A$s with probability* $\frac{1}{14}$.

The choices which lead to the probabilities in the previous lemma are made independently for each trace. Hence, for $n$ traces and a pair 01, there are no $A$s in every trace with probability $(\frac{9}{14})^n$ and one $A$ in every trace with probability $(\frac{5}{14})^n$. A similar result holds for the pair 11. By averaging:

**Lemma 5.** *For the default values of the parameters and $n$ traces, in every trace a bit pair of the form* $*1$ *has:*
*i) no $A$s between the associated $D$s with probability* $\{(\frac{9}{14})^n + (\frac{2}{7})^n\}/2$ *; and*
*ii) one $A$ with probability* $\{(\frac{5}{14})^n + (\frac{9}{14})^n\}/2$.

To reconstruct the key $k$, first classify every bit pair as 00 if there are no intervening $A$s in any trace, 10 if there is always one intervening $A$, 11 if there is an intervening $AA$, and, otherwise, $*1$ if there is a variable number of intervening $A$s. This correctly classifies all pairs 00 and 10, and pairs classed as 11 or $*1$ are certainly all 11 or of the form $*1$ respectively. For $n = 10$ both probabilities in the lemma are bounded above by $\frac{1}{2}(\frac{9}{14})^{10} \approx 1/166$. Thus about 1 in 83 bits pairs 01 and 11 will be incorrectly classified as 00 or 10. Also, by the next lemma, $1-(\frac{6}{7})^{10} > \frac{3}{4}$ of pairs 11 will be located correctly by occurrences of $AA$ when they are the left pair in triplets 111. The proof of it goes back to Lemma 2.

**Lemma 6.** *For the default values of the parameters and $n$ traces, the bit pair* 11 *has at least one trace exhibiting $AA$ with probability* $1-(\frac{6}{7})^n$ *if it has a* 1 *to the right and with probability* 0 *if it has a* 0 *to the right.*

This is now enough information to deduce almost all the bits of a standard length ECC key. Every bit which is deduced as the right member of a pair $*1$ is correctly classified as 1 since the mixture of patterns used in the classification is not possible for pairs of the form $*0$. However, about 1 in 83+1 of the bits which are deduced to be right members of a pair $*0$ is incorrectly classified as 0 because not all the possible patterns for the bit pair have occurred. In an ECC key of, say, 192 bits, about two bits will then be incorrect.

Each bit $b$ belongs to two pairs: $*b$ and $b*$, say. Traces for the pair $*b$ have been used to classify $b$. In half of all cases, there is a 0 bit to the right and the characteristic patterns of traces for the pair $b0$ can be used to cross-check the classification. In the other half of cases the patterns for $b1$ also indicate the correct value for $b$ as a result of the ratios between the numbers of occurrences of each pattern. However, the patterns observed for overlapping bit pairs are not independent. Although unlikely, one set of patterns may reinforce rather than contradict a wrong deduction from the other set. There is no space for further detail, but the following is now clear:

**Theorem 1.** *Suppose elliptic curve adds and doubles can be distinguished accurately on a side channel. If the original Oswald-Aigner exponentiation algorithm is used with the same unblinded* 192*-bit ECC key $k$ for* 10 *point multiplications then approximately half the bits can be deduced unambiguously to be* 1*, and the remaining bits deduced to be* 0 *with an average of at most about two errors.*

This theorem says that a typical ECC secret key can usually be recovered on a first attempt using a dozen traces with very little computational effort beyond extracting the add and double patterns from each trace. By checking consistency between deductions of overlapping bit pairs, most errors should be eliminated. However, it is computationally feasible to test all variants of the deduced key for up to two or three errors. The correct one from this set can surely be established by successfully decrypting some ciphertext.

### 4.4 Secure Parameter Choices?

From the last section, it is clear that greater security could only arise from making it less easy to distinguish between pairs of the form $*0$ and those of the form $*1$. This requires choosing parameters for which 01 and 11 are less likely to exhibit both no $A$s and one $A$ between the relevant $D$s. From Fig. 3, the probability of no $A$s for 01 and the probability of one $A$ for 11 are the same, *viz.*

$$\pi = (\tfrac{1}{2}-\mu)\alpha + (\tfrac{1}{2}-2\mu)\beta + \mu\delta.$$

So this must be made close to 0 or close to 1. For example, choosing $\alpha = \beta = 1$ makes $\mu = 0$ and so $\pi = 1$, whereas choosing $\alpha = \beta = \delta = 0$ makes $\pi = 0$. Thus both limits are possible. In general, for $\pi = 1$ (the first case) the traces match the pattern of operations for normal square-and-multiply, so we expect each $A$ to correspond to the multiply of a 1 bit. Although 00 and 01 are indistinguishable from the patterns, and 10 and 11 are indistinguishable (unless perhaps $AA$ could occur), the attacker now recognises that patterns for the pairs $0*$ have no intervening $A$ and patterns for the pairs $1*$ have one intervening $A$. This gives him each bit unequivocally. At the opposite extreme, if $\pi = 0$ (the second case) then 10 and 01 become indistinguishable from the patterns as do 00 and 11 (again, unless perhaps $AA$ could occur). Now the attacker recognises pairs with equal bits from pairs with different bits. Knowing the first bit is 1, he can deduce all the bits one by one from left to right, and hence the key $k$.

In general the attacker can exploit the complementary frequencies of one $A$ for the pairs 01 and 11. Either they are close enough to ensure $n$ traces usually display both patterns (as in the previous section) or they are distinct enough for the patterns to be strongly biased in opposite directions in the trace set (as in the previous paragraph). He can then recognise either the equality of the second bits or the difference in the first bit respectively, and use the fact that each bit belongs to two pairs to cross-check the deduction of many bits. Consequently, there are no secure choices of the parameters under repeated use of the unblinded key $k$.

Identical working to the previous section shows that similar computations can be performed for keys of any length. With the choice of parameters there, the number of traces needed to achieve a specified degree of confidence in the determined bits is $n = O(\log\log k)$ because we want at most one error in $(\frac{14}{9})^n = O(\log k)$ bits. The same calculations apply for any $\pi$ which is not 0 or 1, giving the same size order for $n$. For the working above in this section, mistakes are only made when too many traces record the opposite pattern to that expected from the value of $\pi$. Then, for $\pi$ close enough to 0 or 1, the same bound on the size of $n$ can be obtained for limiting the errors. So,

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

When adds and doubles are not distinguished with 100% certainty, the proportions of numbers of $A$s can be used to assign a likelihood to the correctness of the selected bit pair. Those which are most likely to be wrong can be modified first, thereby decreasing the search time to determine the correct key.

### 4.5 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 character subsequences shared between their traces. So the size of the random number used in blinding cannot reasonably be less than the maximum lifespan of the key in terms of the number of point multiplications for which it is used. Thus 16 or more bits are needed, adding around 10% to the cost of point multiplication.

Identical formulae for additions and doublings are increasingly efficient and applicable to wider classes of elliptic curves, those of Brier and Joye [1] in particular. These should make it more difficult to distinguish adds from doubles.

Another favoured counter-measure is the add-and-always-double approach. Then the pattern of adds and doubles is not key dependent. Each occurrence of $DD$ has an add inserted to yield the pattern $DAD$, but the add output is discarded without having been used. This can also be done for the Oswald-Aigner algorithm provided, in addition, an extra double is performed to convert each $DAAD$ into $DADAD$. The output of this double is likewise ignored.

Alternatives algorithms exist. That described by Joye and Yen [5] is another add-and-always-double algorithm. There are also several randomized methods [3, 15] which seem to be more robust because they do not satisfy the two properties identified in the introduction as those to which the above attack can be applied.

## 5 One Trace

It is interesting to speculate on how much data leaks from a single point multiplication since the above counter-measures should prevent re-use of identical values for the same key. Oswald [10] noted that for some *deterministic* re-coding algorithms in which several non-zero digits generate indistinguishable $A$s, 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 likelihood of occurrence, and this considerably reduces the average search time for the correct key. Hence the key can be recovered quite easily.

Is the same possible here? In [10] 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 3 above does not provide certainty for many bits unless a number of traces are available; only the infrequent instances of $AA$ seem to allow definite determination of any bits from one trace. Of course, an analysis of sub-sequences of more than two bits is possible, as in [14], but, besides better probabilities, this gives no further insight into whether it is computationally feasible to recover the key from a single trace.

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. Thus, a standard ECC key could be recovered from a single trace in feasible time if adds and doubles are clearly distinguishable.

## 6 Conclusion

One of several, similar, randomized exponentiation algorithms has been investigated to assess its strength against a side channel attack which can differentiate between elliptic curve point additions and point doublings. Straightforward theory shows that at most $O(10)$ uses of the same unblinded key will enable a secret key of standard length 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 one side channel trace is 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 (eds), LNCS **2274**, Springer-Verlag, 2002, pp. 335–345.
2. C. Gebotys & R. Gebotys, *Secure Elliptic Curve Implementations: An Analysis of Resistance to Power-Attacks in a DSP Processor*, CHES 2002, B. Kaliski, Ç. Koç & C. Paar (eds), LNCS **2523**, Springer-Verlag, 2003, pp. 114–128.
3. K. Itoh, J. Yajima, M. Takenaka & N. Torii, *DPA Countermeasures by improving the Window Method*, CHES 2002, B. Kaliski, Ç. Koç & C. Paar (eds), LNCS **2523**, Springer-Verlag, 2003, pp. 303–317.
4. M. Joye & J.-J. Quisquater, *Hessian Elliptic Curves and Side Channel Attacks*, CHES 2001, Ç. Koç, D. Naccache & C. Paar (eds), LNCS **2162**, Springer-Verlag, 2001, pp. 402–410.
5. M. Joye & S.-M. Yen, *The Montgomery Powering Ladder*, CHES 2002, B. Kaliski, Ç. Koç & C. Paar (eds), LNCS **2523**, Springer-Verlag, 2003, pp. 291–302.
6. P. Kocher, *Timing Attack on Implementations of Diffie-Hellman, RSA, DSS, and other Systems*, Advances in Cryptology – Crypto '96, N. Koblitz (ed), LNCS **1109**, Springer-Verlag, 1996, pp. 104–113.
7. P. Kocher, J. Jaffe & B. Jun, *Differential Power Analysis*, Advances in Cryptology – Crypto '99, M. Wiener (ed), LNCS **1666**, Springer-Verlag, 1999, pp. 388–397.
8. P.-Y. Liardet & N. P. Smart, *Preventing SPA/DPA in ECC Systems using the Jacobi Form*, CHES 2001, Ç. Koç, D. Naccache & C. Paar (eds), LNCS **2162**, Springer-Verlag, 2001, pp. 391–401.
9. T. S. Messerges, E. A. Dabbish & R. H. Sloan, *Power Analysis Attacks of Modular Exponentiation in Smartcards*, Proc. CHES 99, C. Paar & Ç. Koç (eds), LNCS **1717**, Springer-Verlag, 1999, pp. 144–157.
10. E. Oswald & M. Aigner, *Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks*, CHES 2001, Ç. Koç, D. Naccache & C. Paar (eds), LNCS **2162**, Springer-Verlag, 2001, pp. 39–50.
11. E. Oswald, *Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems*, CHES 2002, B. Kaliski, Ç. Koç & C. Paar (eds), LNCS **2523**, Springer-Verlag, 2003, pp. 82–97.
12. J.-J. Quisquater & D. Samyde, *ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards*, Smart Card Programming and Security (e-Smart 2001), I. Attali & T. Jensen (eds), LNCS **2140**, Springer-Verlag, 2001, pp. 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, pp. 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 (eds), LNCS **2384**, Springer-Verlag, 2002, pp. 420–435.
15. C. D. Walter, *Mist: An Efficient, Randomized Exponentiation Algorithm for Resisting Power Analysis*, Proc. CT-RSA 2002, B. Preneel (ed), LNCS **2271**, Springer-Verlag, 2002, pp. 53–66.
16. C. D. Walter, *Breaking the Liardet-Smart Randomized Exponentiation Algorithm*, Proc. Cardis '02, San José, November 2002, Usenix Association, Berkeley, 2002, pp. 59–68.