USENIX Association

Proceedings of the 5th Smart Card Research and Advanced Application Conference

San Jose, California, USA
November 21–22, 2002

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Breaking the Liardet-Smart Randomized Exponentiation Algorithm

Colin D. Walter

Comodo Research Laboratory
10 Hey Street, Bradford, BD7 1DQ, UK

colin.walter@comodo.net www.comodo.net

Abstract. In smartcard encryption and signature applications, randomised algorithms are used to increase tamper resistance against attacks based on side channel leakage. Recently several such algorithms have appeared which are suitable for RSA exponentiation and/or ECC point multiplication. We show that under certain apparently reasonable hypotheses about the countermeasures in place and the attacker's monitoring equipment, repeated use of the same secret key with the algorithm of Liardet and Smart is insecure against any side channel which leaks enough data to differentiate between the adds and doubles in a single scalar multiplication. Thus the scalar needs to be blinded in the standard way, or some other suitable counter-measures employed, if the algorithm is to be used safely in such a context.

Key words: *m*-ary exponentiation, Liardet-Smart randomized algorithm, ECC, addition chains, sliding windows, addition-subtraction chains, power analysis, SPA, SEMA, blinding, smartcard.

1 Introduction

Major progress in the theory and practice of side channel attacks [5, 6] on embedded cryptographic systems threatens to enable the capture of secret keys from *single* applications of cryptographic functions [10, 11, 14]. 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.

Timing attacks on modular multiplication can usually be avoided easily by removing data-dependent conditional statements [16], but, with timing variations removed, attacks which make use of data-dependent variation in power and electro-magnetic radiation become easier. Initial attacks of this type required averaging over a number of exponentiations [8].

One counter-measure is to modify the exponent 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 [5]. This results in a different exponentiation being performed every time. However, if squares and multiplications can be distinguished during a single exponentiation, then use of the standard binary exponentiation algorithm immediately leads to exposure of the secret key.

For elliptic curve cryptography (ECC), the most efficient schemes for point addition and point doubling involve different numbers of operations in the field over which the curve is defined, and these numbers vary depending on the representation used for the curve. A counter-measure which reduces the likelihood of distinguishing between these point operations involves equalising the number and type of the component field operations [12] or making the point addition look exactly the same as two point doublings [1].

However, squares and multiplications in the field behave differently [13] and so there is no reason to believe that such recoding will necessarily hide fully the distinction between point additions and point doublings: for example, in [12], field squares appear for point additions, but field cubes when the same formula is used for point doublings. Side channels can distinguish these if the Hamming weight of arguments can be deduced. So exponentiation algorithms are chosen in which there is still an ambiguity in the correspondence between multiplications (i.e. point additions in ECC terms) and properties of the secret key (such as bit or digit values). *M*-ary exponentiation [4] for $m > 2$ provides one solution because each addition represents an unknown choice from a set of several non-zero digits.

If the same unblinded key value is re-used for many exponentiations, there is a danger that the repeated use of the same operand can be determined [14]. This would enable individual digits of the exponent base m to be identified and hence the key recovered. Unfortunately, particularly for ECC as opposed to RSA, applying the above exponent blinding technique is expensive when the secret key is typically only 192 bits. It adds about 17% to the cost of point multiplication. Hence randomised exponentiation algorithms may be a preferred option for ECC.

There are currently several algorithms which randomise the operations associated with specific inputs so that the exponentiation scheme is different on successive runs with the same data [7, 9, 17, 2, 3]. That of Liardet and Smart [7] uses a sliding window of random, variable width. If the attacker's equipment is insufficient to obtain information from a single EC point multiplication, then it seems that averaging over different multiplications with the same key would dilute any data dependency in the side channel leakage. However, we will show here that if individual point multiplications do leak information about what operation is being performed, then the secret key can be obtained straightforwardly. Indeed, one might even be better off with *m*-ary multiplication.

We begin by recalling the algorithm and looking at various parameters which might be chosen to improve efficiency or security. Next, the assumptions about the attacker's equipment and cryptosystem counter-measures are outlined. These are initially quite tight to make the presentation of the attack easier. The attack starts with extracting a least significant digit, and then uses this repeatedly to reconstruct one possible representation for the secret key. An essential part of the discussion is an assessment of the probability that the attack can be completed successfully. Before concluding with some counter-measures and alternatives, we explain how the attack can still be performed in a more realistic environment where the side channel leakage is much poorer.

2 The Algorithm

This section contains a brief outline of the (exponentiation) algorithm of Liardet and Smart. Because it generates an addition-subtraction chain rather than simply an addition chain, inverses have to be computed when it is applied. This means that applications to RSA cryptography are unlikely because of the expense of computing inverses. However, in elliptic curve cryptography (ECC), inverses are essentially for free. Hence, we will assume the algorithm is applied to an additive group, such as that formed by the points on an elliptic curve, and use appropriate terminology. Processing of the secret key k therefore produces a sequence of instructions which result in additions (A) and doublings (D) of group elements.

Suppose we wish to compute the element $Q = kP$ for a given positive integer k (the secret key) and a given member P of some group E. As in m-ary exponentiation, Liardet and Smart pre-compute the odd multiples iP of P for integers $i \in (-\frac{1}{2}m, \frac{1}{2}m]$ where $m = 2^R$, and then employ the standard sliding windows technique but with a window which has a random width showing up to R bits. In other words, k is recoded to obtain digits k_i $(0 \le i \le n)$ which are determined using a randomly-chosen variable base m_i which divides m. The digits are chosen in the order $k_0, k_1, \ldots, k_n$ and the digit representation $k_nk_{n-1}\ldots k_1k_0$ satisfies

$$k = ((\ldots((k_n)m_{n-1}+k_{n-1})m_{n-2}+\ldots)m_1+k_1)m_0+k_0 \qquad (1)$$

The group element multiplication processes these digits from most to least significant following the related scheme defined by

$$kP = m_0(m_1(\ldots m_{n-2}(m_{n-1}(k_nP)+k_{n-1}P)+\ldots)+k_1P)+k_0P \quad (2)$$

2.1 Code for the Key Recoding

More explicitly, if minmod is the function which returns a residue of minimal absolute value, the algorithm for choosing the digits is this:

RANDOMISED SIGNED *m*-ARY WINDOW DECOMPOSITION [7]:

```
i ← 0 ;
While k > 0 do
{     If (k mod 2) = 0 then
      {     $m_i$ ← 2 ;
            $k_i$ ← 0 ;
      }
      else
      {     Randomly choose base $m_i \in \{2^1, 2^2, \ldots, 2^R\}$ ;
            $k_i$ ← k minmod $m_i$ ;
      } ;
      k ← $(k-k_i)/m_i$ ;
      i ← i+1 ;
} ;
```

Here, both 1 and $\bar{1}$ could be allowed as digits for base 1, but that involves the added complication of a random bit to decide which to select, and also (to avoid non-termination) restricting the choice to only 1 when k reaches 1. Our attack would work also in these circumstances with few changes.

2.2 Efficiency Considerations

There are still some parameters to be chosen in the algorithm. Varying these affects efficiency, but there are also security implications. As we see later, certain choices will increase the difficulty of mounting the attack, forcing, in particular, more samples to be required.

The value for R has the greatest effect on efficiency. In elliptic curve applications, subtraction may have the same cost as addition. Then it will be unnecessary to store the negative pre-computed multiples of the input point. So only space for 2^R-2 multiples is likely to be required. Increasing R improves speed, but with diminishing returns for the space required for pre-computed values.

No suggestions are made in [7] about how to choose the m_i randomly. A uniform distribution is not very efficient, and indeed perhaps the least secure under this attack. It is most efficient to make the maximum possible use of the pre-computed values by choosing the maximum base size 2^R always. But, to maintain generality for later, suppose $m_i = 2^j$ is chosen with probability p_j when k is odd and $p_0 = 1$ is the probability of selecting base 2 when k is even.

Choosing $p_R = 1$ means that $m_i = 2^R$ whenever k is odd. This yields the usual m-ary sliding window method with fixed $m = 2^R$. Taking $R = 1$ yields the usual binary "square-and-multiply" algorithm. However, such choices would remove any non-determinism from the sequence of point operations.

Observe that biasing in the choice of m_i does not change the uniformity in the distribution of residues k mod m_i inherited from k, assuming k is randomly chosen. This means that every new key value k generated during the recoding retains the same random properties: in particular, residues modulo 2^j will be uniform for every key encountered.

3 The Attack

3.1 Introduction & Initial Hypotheses

The purpose of randomised 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 might soon be breached without excessive expenditure [10, 11].

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 multiplications. 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.

Randomisation 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. Here every point multiplication determines a different sequence of doublings and additions. With matched code for additions and doublings, 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 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 different scalar multiplications.

Both of these hypotheses will be relaxed later to some extent, providing a more realistic scenario.

3.2 Overview of the Attack & Notation

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

- Every trace can be viewed as a word over the alphabet $\{A,D\}$.

Every occurrence of an A (add) in the trace splits the word into a prefix and a suffix which correspond to two integers k_p and k_s that are precisely defined in terms of k

and the position of A in the trace. All traces determine the same values to within ±1. By looking at the patterns to the left of a given A, one obtains the residue of k_p modulo a small power of 2, and hence a few bits of k. Repeating this for the position of each A enables all the bits of k to be recovered.

Definition. *The **position** of a specific instance of character A or D in a trace word is the number of Ds which are to the right of the selected character.*

We will exploit a close relationship between positions in which A appears in traces and bits which are 1 in the corresponding position of the binary representation of k.

In order to be able to give examples, we fix the character order of words over $\{A,D\}$ to correspond to a left-to-right processing order. Thus, the digit sequence $1_2 0_2 \bar{1}_4 0_8$, with most significant bit on the left, is processed from most to least significant bit, i.e. from left to right, and so would result in the word *DADDDADD*. There are As in positions 2 and 5, and Ds in positions 0 to 5. In fact, every A is paired with a preceding D with the same position, and so one could view the DA combination as a single character. Then the position would correspond directly to a character index, counting from 0 at the right hand end, as in a binary representation.

The initial DA corresponds to the digit 1_2 and might be omitted if efficient initialisation takes place instead. Assuming this is the case, we will delete any initial Ds but leave the initial A as an unambiguous reminder that there is an initial digit to take into account. Thus words always commence with A. In the above example, *ADDDADD* will be the word corresponding to the given digit sequence.

3.3 Properties of Key Digits

We now look at the sequence of digits generated by the Liardet-Smart algorithm. The notation used here is the same as for a fixed base, and many standard properties have analogues.

Lemma 1. *Suppose $k_n k_{n-1} \ldots k_1 k_0$ is a digit representation of k generated by the Liardet-Smart algorithm, with sequence $m_n, m_{n-1}, \ldots, m_1, m_0$ of bases. For some i, let $k_p^{(i)}$ denote the integer corresponding to the prefix $k_n k_{n-1} \ldots k_i$ and let $k_s^{(i)}$ denote the integer corresponding to the suffix $k_{i-1} \ldots k_1 k_0$. Then $k = k_p^{(i)} m^{(i)} + k_s^{(i)}$ where $m^{(i)} = \prod_{j=0}^{i-1} m_j$ and $|k_s^{(i)}| < m^{(i)}$.*

This lemma is obvious from the definition of the digit sequence given by equation (1) except, perhaps, for the last part. That part follows easily by induction. Digit k_i is chosen with $|k_i| \le ½m_i \le m_i - 1$. This property for $i = 0$ starts the induction. Then the induction step is
$$|k_s^{(i+1)}| = |k_i m^{(i)} + k_s^{(i)}| < |(k_i+1) m^{(i)}| \le m_i m^{(i)} = m^{(i+1)}.$$

We will continue to use the notation $m^{(i)}$ for $\prod_{j=0}^{i-1} m_j$ and $k_p^{(i)}$ and $k_s^{(i)}$ for the key values associated respectively with the prefix $k_n k_{n-1} \ldots k_i$ and the suffix $k_{i-1} \ldots k_1 k_0$ of the digit sequence. The next lemma uses the equality and bound of Lemma 1 to identify two possible values for $k_s^{(i)}$, corresponding to it being a positive or negative residue of k modulo $m^{(i)}$:

Lemma 2. *With the previous notation, either*

i) $k_s^{(i)} = k \bmod m^{(i)}$ *and* $k_p^{(i)} = k \operatorname{div} m^{(i)}$, *or*

ii) $k_s^{(i)} = (k \bmod m^{(i)}) - m^{(i)}$ *and* $k_p^{(i)} = (k \operatorname{div} m^{(i)}) + 1$

where mod *returns the least non-negative residue, and* div *is integer division given by rounding down the real quotient.*

This shows that, whatever choices are made for the base elements, a given digit suffix can determine only one of two possible values when the product of the corresponding base elements is fixed. We would like at least the occurrence of the one which will make the corresponding prefix odd because it leads to an addition (A) which can be used to identify a corresponding point in the trace.

Lemma 3. *For all powers $2^j \le k$, there is a choice of base elements $m_{i'}$ and an integer i such that $2^j = m^{(i)}$. On average, for at least $1-2^{-R}$ of all values of j, there are choices which make $k_p^{(i)}$ odd, and, for at least half of all values of j, there are choices which make $k_p^{(i)}$ even.*

Proof. The existence of the choice of basis elements is clear: taking $m_{i'} = 2$ for all i' allows one to satisfy the equality for $m^{(i)}$ with $i = j$. For that choice, k_i is the usual index i bit of k, and takes either parity with equal probability. It is the lowest bit of $k_p^{(i)}$, and so $k_p^{(i)}$ is the desired parity with probability ½.

Other choices of base elements exist, and they may result in $k_p^{(i)}$ being odd even when the bit of interest in k is even. This increases the average number of cases for which oddness occurs. Instead of choosing $m_{j-1} = 2$, we try choosing $m_{j-i'} = 2^{i'}$ for any i' with $1 \le i' \le R$. The ability to select them depends on a corresponding bit of $k_p^{(j-R)}$ being 1 (otherwise base 2 must be chosen). These bits are independent and so the alternative bases can be chosen with probability ½. Each will give odd parity to the next k_p if chosen. Hence the prefix key corresponding to 2^j can be made odd with probability at least $1-2^{-R}$. ■

Of course, this argument just gives a lower bound on how many js will give rise to two key prefix and two suffix values. It doesn't guarantee that when both values are possible they will both appear with non-negligible frequencies. The actual relative frequencies appear to depend on the lowest R bits of the k_p corresponding to $m^{(j)} = 2^{j-R}$. However, in the next section, a lower bound on the ratio will be produced as necessary for each choice of these bits.

3.4 Recovering One Digit of *k*

In this section we show how to recover the least significant digit k_0 and associated base m_0 in one representation of k and how to identify the subset of traces which correspond to the associated prefix key k_p such that $k = k_p m_0 + k_0$. Exactly the same process yields other digits of k independently. Those digits can then be assembled together to give k in the manner described in the next section.

If Tr is the full set of all sample traces, then we denote by Tr_i the set of traces obtained by taking each member of Tr and deleting the suffix to the right of, but not including, the D of position i. Thus $Tr_0 = Tr$. Tr_i is partitioned into two complementary subsets: Tr_i^A which consists of those traces which terminate with A, and Tr_i^D which consists of those traces which terminate with D. We need to identify one of these subsets for each digit choice so that its neighbour to the left can be selected correctly. Tr_i^A always represents the odd choice for $k_p^{(i)}$, but some traces in Tr_i^D may contain only some of the operations for the rightmost prefix digit, and so not represent any $k_p^{(i)}$ properly.

The derivation here does make specific use of the fact that in this implementation $\bar{1}$ is not allowed as a digit for base 2. Similar arguments apply when $\bar{1}$ is allowed, but there is a duality which leaves a complicating ambiguity between the two values of $\pm k_r$ throughout the reconstruction process. This is only resolved when the complete value of k is reconstructed and under the assumption that the true sign of k is known.

Lemma 4. *Select any trace for key k. Then k is exactly divisible by* 2^i *where i is the uniquely defined integer such that* AD^i *is a suffix of the trace.*

Proof. Clearly, if k is divisible by 2^i then base 2 must be chosen for the lowest i digits, which are then all zeros. This leads to a character sequence D^i of i consecutive Ds as a suffix in every trace. If k is not divisible by 2^{i+1} then, whatever the next choice of base, the digit will be non-zero and hence cause A to be appended to the sequence, yielding the suffix AD^i. ■

This result enables these i occurrences of D to be identified with i least significant digits 0, each of base 2. Moreover, all traces confirm this conclusion. So, removing the digits one at a time,

Lemma 5. *If every trace in Tr has final character D then we may take* $k_0 = 0$, $m_0 = 2$ *and the traces of* Tr_1 *all represent the associated* k_p.

If k is odd, no digit has been deduced yet, and further work must be done.

Lemma 6. *Suppose* $k \equiv 1 \bmod 2^i$ *where* $i \le R$. *Then* $k \equiv 2^i{+}1 \bmod 2^{i+1}$ *if Tr contains a trace with suffix* AD^iA. *If* $k \equiv 2^i{+}1 \bmod 2^{i+1}$ *then the probability that Tr contains no trace with suffix* AD^iA *is* $(1-p_i')^{|Tr|}$ *where* $p_i' = p_1+p_2+\ldots+p_i$.

Proof. If $k \equiv 1 \bmod 2^{i+1}$ then a base of $m_0 = 2^{i+1}$ or larger will lead to suffix $D^{i+1}A$. However, a smaller base $m_0 = 2^j$ will lead to suffix D^jA with digit $k_0 = 1$ and the forced selection of base 2 at least $i{+}1{-}j$ times. This again leads to suffix $D^{i+1}A$.

Now suppose $k \equiv 2^i{+}1 \bmod 2^{i+1}$. A base of $m_0 = 2^{i+1}$ or larger again leads to suffix $D^{i+1}A$. However, the choice of base $m_0 = 2^i$ means lowest digit $k_0 = 1$ and next digit determined by k div 2^i, which is odd. Hence the suffix is AD^iA for that choice. Similarly, a base $m_0 = 2^j$ with $j < i$, will lead to suffix D^jA, digit $k_0 = 1$ and $k_p \equiv 2^{i-j} \bmod 2^{i-j+1}$. So this choice is followed by the forced selection of base 2 exactly $i{-}j$ times with associated digit 0. The subsequent digit is then odd, resulting in the overall suffix AD^iA.

Thus, suffix AD^iA guarantees $k \equiv 2^i{+}1 \bmod 2^{i+1}$ and it occurs precisely when the least significant base choice is 2^j with $j \le i$. These choices occur for a given trace with probability $p_i' = p_1+p_2+\ldots+p_i$. Hence suffix AD^iA will not happen for any trace in Tr with probability $(1-p_i')^{|Tr|}$. ■

Lemma 7. *Suppose* $k \equiv 1 \bmod 2^i$. *If Tr contains a trace with suffix* AD^iA *then* $k \equiv 2^i{+}1 \bmod 2^{i+1}$, *we may take* $k_0 = 1$ *and* $m_0 = 2^i$, *and the traces of* Tr_i^A *all represent the associated* k_p.

This lemma deals with the recognisable instances of $k \equiv 2^i+1 \bmod 2^{i+1}$. When base 2^j is chosen for any $j \le i$, the suffix is AD^jA for these cases. As this occurs in p_i' of cases, so we expect Tr_i^A to contain approximately $p_i'|Tr|$ elements.

We will assume $k \equiv 1 \bmod 2^{i+1}$ if there is no suffix AD^iA but we know $k \equiv 1 \bmod 2^i$. By Lemma 6, this introduces a small probability of error which can be decreased by taking a larger sample if necessary, or by further analysis, such as through a more exhaustive analysis of suffixes and their expected frequencies than there is space for here. Note, however, that if $p_i' = 0$ then this choice of m_0 will not resolve which residue mod 2^{i+1} is correct. Hence an increase in security might be obtained by having $p_1 = p_2 = ... = p_i = 0$ where i is as large as possible.

Theorem 1. *Assume each base* 2^i *is selected with probability* p_i *for odd key values, and digit* $\bar{1}$ *is only used for bases greater than 2. Let* $p_i' = p_1+p_2+...+p_i$ *and* $\overline{p_i'} = 1-p_i'$. *Suppose k is a random odd integer that has generated trace set Tr and j* $(1 \le j \le R+1)$ *is such that Tr contains no trace with suffix* AD^iA *for any* $i < j$. *Then* $k \equiv 1 \bmod 2^j$ *with probability* $\prod_{i=1}^{j-1}(1+\overline{p_i'}^{|Tr|})^{-1}$.

Proof. We prove this by induction on j. For $j = 1$ the statement claims nothing, and so holds. For the induction step, assume the statement holds for some $j \le R$. Suppose also that Tr contains no trace with suffix AD^iA for any $i \le j$. By the induction hypothesis, $k \equiv 1 \bmod 2^j$ with probability $\prod_{i=1}^{j-1}(1+\overline{p_i'}^{|Tr|})^{-1}$.

Since k is random, the two possibilities for $k \bmod 2^{j+1}$ are equally likely. So, by Lemma 6, no occurrence of suffix AD^jA means $k \equiv 1 \bmod 2^{j+1}$ with probability $(1+\overline{p_j'}^{|Tr|})^{-1}$. This factor just needs multiplying into the product to obtain the claim for $j+1$ in place of j. ■

Theorem 2. *With assumptions and notation as in Theorem 1, suppose k is odd and j is minimal such that* $1 \le j \le R+1$ *and Tr contains a trace with suffix* AD^jA. *Then* $k \equiv 2^j+1 \bmod 2^{j+1}$ *with probability*

$$\prod_{i=1}^{j-1}(1+\overline{p_i'}^{|Tr|})^{-1} .$$

If $j \le R$ *we may take* $k_0 = 1$ *and* $m_0 = 2^j$ *and then the set of traces for the associated* k_p *is* Tr_j^A.

Proof. Theorem 1 shows that, for the given definition of j, $k \equiv 1 \bmod 2^j$ with probability $\prod_{i=1}^{j-1}(1+\overline{p_i'}^{|Tr|})^{-1}$. If $k \equiv 1 \bmod 2^{j+1}$ then, as in the proof of Lemma 6, all traces must terminate with suffix $D^{j+1}A$, which is not the case. Hence $k \equiv 2^j+1 \bmod 2^{j+1}$ with the stated probability.

For the base $m_0 = 2^j$, $k \equiv 1 \bmod m_0$ and so the associated digit is $k_0 = 1$. However, $k_p = k \text{ div } 2^j$ is odd, which forces the next digit to be non-zero. Hence A is the next operation leftwards after the suffix D^jA which corresponds to m_0. Thus, the relevant traces for the next digit are those of Tr_j^A. ■

The values of k for which no least significant digit has yet been assigned are those satisfying $k \equiv 1 \bmod 2^{R+1}$. Picking maximal base $m_0 = 2^R$ gives $k_0 = 1$ and makes k_p even. The associated set of prefix traces should be Tr_R^D. A possible difficulty with this definition is that for some traces removing the suffix D^RA may split subsequences which correspond to one digit. However, every choice of base 2^i corresponds to a suffix D^iA where $i \le R$, and must be followed by a number of instances of base 2 with digit 0 which makes the total modular division by at least 2^{R+1}. Hence the suffix D^RA corresponds to the operations for a whole number of digits. Therefore Tr_R^D does indeed contain traces which represent only operations for sequences of complete digits, and so those traces all represent the same key value.

Theorem 3. *With the same assumptions and notation as in Theorem 1, suppose every trace in Tr has suffix* $D^{R+1}A$. *Then, with probability* $\prod_{i=1}^{R}(1+\overline{p_i'}^{|Tr|})^{-1}$, $k \equiv 1 \bmod 2^{R+1}$ *and we may pick* $m_0 = 2^R$. *For this choice* $k_0 = 1$, k_p *is the common key for* Tr_R^D, k_p *is even, and* $Tr_R^D = Tr_R$ *has the same cardinality as Tr.*

3.5 Combining Digits to Recover *k*

For every position j at which there is an occurrence of A in some trace of Tr, the procedures of the previous subsection can be applied to Tr_j^A to obtain a base and digit at that point. These digits are used when determining a digit sequence for k. Starting at $j = 0$, the digits are selected iteratively. As well as a digit and base, each trace set Tr_j^A gives rise to another trace set defined at some position $j' > j$. We will show that:

- For this definition of j', the next digit is determined by whichever is appropriate of $Tr_{j'}^A$ or $(Tr_j^A)_R^D$.

Here we need to check on the definition of the trace subsets. If applied iteratively, the procedures above would actually determine smaller and smaller subsets: each time we apparently take a subset of the traces from the previous step. However, because only two key values (one odd, one even) are associated with any position, every prefix which represents the operations of a complete number of digits must correspond to the odd key if it terminates with A and the even key if it terminates with D. Of course, every trace prefix terminating with A must consist of the operations for a whole number of digits since A cannot appear in the middle of the sequence of operations for a single digit. So every trace in Tr_j^A is generated from a key value which is common to them all. Hence, the full set Tr_j^A can be used to determine the next digit, not just the subset of Tr_j^A determined by the procedures above.

In the case of the prefix trace set Tr_{j+R}^D, it is not clear which traces are generated by a complete key. In some cases, the final D may not be the final operation of the digit sequence from which it was derived. Hence, the subset $(Tr_j^A)_R^D$ must be used, not Tr_{j+R}^D. However, the construction observed that every such trace had suffix $D^{R+1}A$. So $(Tr_j^A)_R^D$ has the same cardinality as Tr_j^A. Hence the trace subsets bulletted above are indeed the correct ones to use for the key digits, and they do not progressively decrease in size.

The process of digit determination only begins to fail once a leading instance of A is encountered: Theorem 2 guarantees progress up to that point. Traces are not all the same length. Some will use a large base for the most significant digit. Their initial Ds are deleted, giving them fewer instances of D overall, making their traces shorter. These traces are simply discarded when fully processed. The procedures above still apply to the subset. Again, following Theorem 2, further digits can still be defined until the trace set becomes empty. However, once the first (i.e. shortest) traces run out, the remaining key is representable by a single digit, so it is bounded in absolute value by $2^{R-1}-1$. Each increment of the position in the trace set reduces the representable key by a factor of 2. Eventually, assuming there are enough traces, the initial A of the longest trace has a digit bounded by 1, and so must be 1. Hence k is completely determined. "Enough" traces would be present if, for example, base 2 were chosen for the most significant digit. Insufficient traces just increases the number of possible values of k which may need testing by a small factor (under 2^{R-1}).

3.6 The Probability of Error

We have been careful to obtain the probability of error in each digit in order i) to see if it is feasible to recover the key and ii) to see how the probabilities p_i might be adjusted in the algorithm definition to provide improved security.

The procedures of §3.4 define the probabilities in terms of the size of the trace set being employed at that time. Generally, it is equal to the cardinality of a set of the form Tr_j^A. This is equal to $|Tr|$ times the number of DAs in position j divided by the number of DAs or Ds in that position. This can be approximated by $|Tr|$ times the overall probability p_A of DA divided by the overall probability p_D of DA or D. Since the choice of base $m = 2^i$ produces $i-1$ occurrences of D followed by one of DA when $i > 0$, $|Tr_j^A| \approx \pi\,|Tr|$ where

$$\pi = \frac{p_A}{p_D} = \frac{p_1+p_2+\dots+p_R}{p_0+p_1+2p_2+\dots+Rp_R}$$

For a uniform distribution this works out at $\pi = \frac{2}{R+3}$ where typically we might expect $R = 3$; and for 2^R-ary sliding windows it works out at $\pi = \frac{1}{R+1}$.

In fact, the formula under-estimates the average size of Tr_j^A. Some positions do not have any occurrences of A, and we do not use the associated trace subsets. This increases the average for those positions which do have occurrences of A.

Next, the distribution of base choices in the reconstructed key differs from that generated by the re-coding process. Suppose k is odd for the set of traces at some point during the reconstruction. In Theorem 2, the distribution of odd residues k mod 2^{R+1} is uniform. So, neglecting the assumed small numbers of incorrectly assigned cases resulting from some of the possible suffixes not occurring, base 2^j will be selected for the reconstructed key with probability 2^{-j} for $0 < j \le R$ and produce an odd next key. Further, base 2^R will turn up in the remaining 2^{-R} cases of odd keys but produce an even next key. In half of all cases, an even key will lead to an even key. Consequently, out of every 2^R+2 digit choices in the reconstruction, on average 2^R will be odd and 2 will be even.

By Theorems 2 and 3, the probability of the reconstructed key being correct is a product of factors of the form $(1+\overline{p_i}^{\,i|Tr_j^A|})^{-1}$. These factors can be approximated by $(1+\overline{p_i}^{\,i\pi|Tr|})^{-1}$. Since choosing base 2^j leads to j such factors, there is essentially one such factor for every bit of k. The exceptions are where an even key causes base 2 and digit 0. Then there is no doubt about the correctness of the digit 0. This last case occurs for $2(2^R+2)^{-1}\log_2 k$ bits of the initial key k. Otherwise, for odd keys, the relative frequency of different bases means that the factor $(1+\overline{p_i}^{\,i\pi|Tr|})^{-1}$ will appear on average for $2^{R-i}(2^R+2)^{-1}\log_2 k$ bits if $0 < i < R$ and for $2(2^R+2)^{-1}\log_2 k$ bits if $i = R$.

Because $p_R' = p_1+p_2+...+p_R = 1$, the factor for $i = R$ is 1, and so can be ignored. Hence,

Lemma 8. *The key k can be recovered with a probability approximately*

$$\prod_{i=1}^{R-1}(1+\overline{p_i}'^{\,\pi|Tr|})^{-2^{-i}n}$$

where $n = (1+2^{1-R})^{-1}\log_2 k$ and π is as defined above.

The property $\overline{p_i}' \leq 1-p_1$ provides a lower bound for this product. Consequently,

Lemma 9. *For a uniform distribution of base, the key k can be recovered with a probability at least*

$$(1+(\tfrac{R-1}{R})^{\pi|Tr|})^{-n}$$

where $n = (1-2^{1-R})(1+2^{1-R})^{-1}\log_2 k$ and $\pi = \frac{2}{R+3}$.

For specific choices, it is possible to evaluate the product in Lemma 8 exactly. A typical choice would be to have a key with 192 bits, $R = 3$ (which requires storing two pre-computed multiples, namely P and $3P$), and a uniform choice of base, i.e. $p_1 = p_2 = p_3 = ⅓$. Then $\pi = ⅓$ and the product in Lemma 8 is just under 2^{-31} for $|Tr| = 9$. So, if a key can be reconstructed and checked for correctness in unit time,

Theorem 4. *If doubles and adds can be distinguished on individual traces, and traces are captured from 9 applications of the same unblinded 192-bit key, then the Liardet-Smart algorithm with uniform selection of base $\leq 2^3$ can be broken with a computational effort of about $O(2^{31})$. With twice as many traces, the computational effort falls to under $O(2^{11})$.*

Of course, the full force of all the patterns available and their relative frequencies has not yet been applied. Hence the danger is probably substantially under-estimated. Once a possible key has been recovered, there is considerable unused data in the traces that has not yet been used and can be investigated for checking purposes. In the uniform case, about $\frac{R+1}{R+3}$ of the data is so far unused – that in the complementary sets Tr_j^D. This contains information about digits whose bases were not aligned with those of the reconstructed representation of k. Choosing a different base from that of the reconstruction process described above will provide confirmation about the correctness of each bit of k. Indeed, each trace has to be consistent with some choice of bases, and the rightmost inconsistency in a trace will usually be very close to the rightmost bit in error. There is insufficient space here to improve the probabilities which are a consequence of this approach, but the computational feasibility of the attack is already assured.

If the attacker is unable to distinguish clearly between adds and doubles, then the unused data vastly increases his ability to make corrections. Moreover, as each digit is obtained through a purely local extraction of data from traces, it is easy to automate an exhaustive process to check for the overall best digit solutions using all traces, and hence prioritise the order for considering the most likely values for k. However, for the data that has been used, any indistinctness between A and D is unimportant. In this attack, it is only necessary to establish whether or not an A has appeared at each position. The relative frequency of As means that the certainty of this can be determined with high degree just by increasing the number of traces sufficiently.

4 Counter-Measures

Our formulae for bounding the accuracy repeatedly used the probabilities of smaller bases much more than larger bases, and the accuracy improves when these probabilities are increased at the expense of the probabilities of larger bases. This is consistent with the greater ambiguity afforded by digits of larger bases. Thus we recommend not using a uniform choice for the base, but employing a strong bias towards large bases, such as was illustrated in §2.2. In the extreme, the standard, non-randomised, m-ary exponentiation technique is obtained, and this is not susceptible to the attack.

The cost of key masking is not entirely trivial in the context of ECC. Adding a 32-bit random multiple of the group order to the key increases the point multiplication cost by some 17% for 192-bit keys, although it is a much smaller fraction of the total encryption cost. Adding a smaller random multiple is probably ineffective if it results in a number of repetitions of the same key value within the lifetime of the key. The highly repetitive nature of the traces resulting from the same prefix keys turning up again and again means that a duplicated key could be assumed if, and only if, traces matched closely enough.

The "double-and-add-always" method of computation provides a good measure of protection, but is expensive. The attacker then has to determine whether or not the result of the addition is used before he can mount the attack. This is much more difficult than distinguishing the two operations. Hence traces will be susceptible to much more frequent errors, and a much greater number of traces will have to be recovered.

There are alternative randomised algorithms for which this type of attack does not apply, and others that display similar weaknesses. That of Oswald and Aigner [9] can be attacked in a similar way. MIST [15, 17] does not exhibit the same repetition of key values during key processing, and so may be a safer choice. A new algorithm by Itoh *et al.* [18] may also be worthy of consideration.

5 Conclusion

It might have been hoped that the Liardet-Smart algorithm would avoid the cost of any additional counter-measures such as key blinding when the same secret key is repeatedly re-used, but this now appears not to be so. Specifically, the key needs to be masked, or the pattern of adds and doubles has to be well hidden for individual point multiplications.

Of course, there are many circumstances in which the algorithm is clearly of value, such as ECDSA, for which a different random key is used every time. Then, for suitable parameter choices, the space of keys generating a given pattern of adds and doubles is infeasibly large, and so cannot be attacked successfully without additional data.

References

[1] C. H. Gebotys & R. J. Gebotys, *Secure Elliptic Curve Implementations: An Analysis of Resistance to Power Attacks in a DSP Processor*, Cryptographic Hardware and Embedded Systems – CHES 2002, B. Kaliski, Ç. Koç & C. Paar (editors), Lecture Notes in Computer Science, **2523**, Springer-Verlag, 2002, *to appear*.

[2] J. C. Ha & S. J. Moon, *Randomized Signed-Scalar Multiplication of ECC to Resist Power Attacks*, Cryptographic Hardware and Embedded Systems – CHES 2002, B. Kaliski, Ç. Koç & C. Paar (editors), Lecture Notes in Computer Science, **2523**, Springer-Verlag, 2002, *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, B. Kaliski, Ç. Koç & C. Paar (editors), Lecture Notes in Computer Science, **2523**, Springer-Verlag, 2002, *to appear*.

[4] D. E. Knuth, *The Art of Computer Programming*, vol. 2, "Seminumerical Algorithms", 2nd Edition, Addison-Wesley, 1981, 441–466.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] J.-J. Quisquater & D. Samyde, *Eddy current for Magnetic Analysis with Active Sensor*, Smart Card Programming and Security (E-smart 2002), Lecture Notes in Computer Science, Springer-Verlag, 2002, *to appear*.

[12] 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.

[13] 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.

[14] 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.

[15] C. D. Walter, *Improvements in, and relating to, Cryptographic Methods and Apparatus*, UK Patent Application 0126317.7, Comodo Research Laboratory, 2001.

[16] C. D. Walter, *Precise Bounds for Montgomery Modular Multiplication and Some Potentially Insecure RSA Moduli*, Proceedings of CT-RSA 2002, Lecture Notes in Computer Science, **2271**, Springer-Verlag, 2002, 30–39.

[17] 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.

[18] K. Itoh, J. Yajima, M. Takenaka & N. Torii, *DPA Countermeasures by Improving the Window Method*, Cryptographic Hardware and Embedded Systems – CHES 2002, B. Kaliski, Ç. Koç & C. Paar (editors), Lecture Notes in Computer Science, **2523**, Springer-Verlag, 2002, *to appear*.