Further Hidden Markov Model Cryptanalysis

P.J. Green[1], R. Noad[2], and N.P. Smart[2]

[1] Department of Mathematics, University of Bristol,
University Walk, Bristol, BS8 1TW, United Kingdom
P.J.Green@bristol.ac.uk
[2] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom
{noad, nigel}@cs.bris.ac.uk

Abstract. We extend the model of Karlof and Wagner for modelling side channel attacks via Input Driven Hidden Markov Models (IDHMM) to the case where not every state corresponds to a single observable symbol. This allows us to examine algorithms where errors in measurements can occur between sub-operations, e.g. there may be an error probability of distinguishing an add (A) versus a double (D) for an elliptic curve system. The prior work of Karlof and Wagner would assume the error was between distinguishing an add-double (AD) versus a double (D). Our model also allows the modelling of unknown values, where one is unable to determine whether a given observable is add or double, and is the first model to allow one to analyse incomplete traces. Hence, our extension allows a more realistic modelling of real side channel attacks. In addition we look at additional heuristic approaches to combine multiple traces together so as to deduce further information.

1 Introduction

The randomization of algorithms as a technique to prevent side channel analysis has in recent years been a topic of intense research. However, many of the approaches using randomization have been made in an ad-hoc manner with little analysis as to whether the randomization introduced actually helps reduce the risk of side channel attacks.

As a trivial example of such a randomization consider the following two variants of the right-to-left binary exponentiation algorithm in an additive group. One is the standard, non-randomized, version whilst the second is a randomized version requiring a random coin per key symbol (usually a bit).

Non-Randomized Binary Method

$Q \leftarrow \mathcal{O}$
$T \leftarrow P$
For $i = 1$ to N
  If $(k_i = 1)$ $Q \leftarrow Q + T$
  $T \leftarrow 2T$
Return Q

Randomized Binary Method

$Q \leftarrow \mathcal{O}$
$T \leftarrow P$
For $i = 1$ to N
  If $(k_i = 1)$ $Q \leftarrow Q + T$
  Else if $(\text{coin}_i = 0)$ $R \leftarrow Q + T$
  $T \leftarrow 2T$
Return Q

J.R. Rao and B. Sunar (Eds.): CHES 2005, LNCS 3659, pp. 61–74, 2005.

Various authors [4,8,6] have suggested the use of finite state machines or Hidden Markov Models (HMMs) as a mechanism to analyse the benefit of randomization techniques in side channel analysis. See [7] for an introduction and partial survey of the application of HMMs to side channel analysis. The idea is that the hidden states of the Markov Model represent the states of the algorithm implementing the countermeasure, whilst the observations represent the side channel itself.

In [4] Karlof and Wagner introduce a concept called an Input Driven Hidden Markov Model, this is a HMM which for each state of the algorithm associates an input state which drives the state transition. This input state is used to model the input of the fixed key to the algorithm. The internal state transitions not only depend on the current state and the random tape given to the algorithm, but also the key symbols. We note that this is only a notational simplification since the inputs in the Karlof/Wagner model can be modelled in a standard HMM by extending the state space of a standard HMM to consist of not only the state of the algorithm but also the corresponding key symbol.

The major innovation of the Karlof and Wagner approach was to allow the modelling of attacks involving multiple traces and the modelling of errors in the state measurements. However, a major drawback was that each internal state and observable had to correspond to a single key symbol. To see the advantages and the disadvantages of this approach we now give an overview of the approach taken by Karlof and Wagner, as applied to randomized group exponentiation algorithms, which are after all the main application area of side channel analysis on public key algorithms.

We shall adopt additive group notation, as is common in elliptic curve cryptography. Suppose we wish to compute $Q = kP$ for a fixed secret integer k and a (possibly fixed) public group element P. Almost all algorithms process the bits of k in chunks (e.g. a bit at a time, or in fixed/sliding window segments). In almost all algorithms the processing of each bit can be reduced to the computation of either a double D, an add-double AD or a double-add DA. Whether one has AD or DA depends on the precise group exponentiation algorithm used, for ease of explanation we shall suppose the algorithm either performs a D or an AD. In simple side channel analysis whether a D or an AD is performed can be deduced from the side channel, the observed sequence of D's and AD's we call a *trace*. Hence, the goal of the attacker is to deduce the value of k given the sequence of D's and AD's observed.

In practice, however, one may not be able to determine correctly the sequence of D's and AD's from the side channel, one may make errors in this observation, or in fact be unable for certain measurements to determine whether a given symbol is A or D. In addition since one is assuming a randomized exponentiation algorithm the attacker could repeat the side channel experiment, assuming the same k is used on each exponentiation (but not necessarily the same P). The attacker then needs to combine the information obtained from multiple traces in such a way as to obtain the secret k.

Karlof and Wagner propose a heuristic method, which they describe as a variant of the Viterbi algorithm although it is actually a variant of the Forward-

Backward (FB) algorithm, to solve the above problem. Using a prior probability distribution on the key symbols (say for example each bit is equally likely to be zero or one), they use their FB algorithm and a single trace so as to deduce an approximation to the posterior probability distribution on the key symbols. This posterior distribution is then used as the prior distribution for the next trace to be processed and so on. After the processing of all of the set of multiple traces one deduces the final estimated posterior distribution on the key symbols, which the attacker hopes reveals to him the actual key used. This form of belief propagation reduces an exponential increase in the number of states needed to process all traces in a parallel manner. However, it is clearly susceptible to the order in which the traces are fed into the algorithm and it does not work for some exponentiation algorithms.

A practical problem with the Karlof and Wagner approach is that each observable, i.e. D or AD, needs to correspond to a single key bit. This is fine when dealing with noiseless data; however, in measuring a power trace it is unlikely that we confuse a D with an AD since they take a significantly different period of time. It is far more likely that we confuse a single D with a single A, and vice versa, or be unable to distinguish a D from an A at all.

To see the problems that the model of Karlof and Wagner can produce, suppose we used a non-randomized right-to-left binary exponentiation algorithm, as above. Now if we saw the sequence DAD then the Karlof and Wagner algorithm would interpret this as a key with two bits. Since the output D corresponds to one bit, whilst the output AD corresponds to another. However, it could be that the measuring equipment mistook the A for a D and that the actual sequence executed was DDD, in other words a key with three bits. Hence, one can see that the error model of Karlof and Wagner does not fully model the errors that one can see in a real-life side channel measurement.

The main motivation for looking at these techniques is to handle situations where one is unable to accurately distinguish an A from a D. This happens when analysing exponentiation algorithms which have been implemented using arithmetic techniques which aim to make the distinguishing of an A from a D as difficult as possible. Such techniques have been proposed by various authors in particular Brier, Déchéne, Joye, Liardet, Quisquater and Smart [2,3,5].

In this paper we extend the Karlof and Wagner approach to cope with traces for which each key symbol potentially corresponds to multiple, or zero, observable symbols and where some of the observable symbols may be unknown or wrong. In particular we model the case where multiple runs of the exponentiation algorithm, with the same secret exponent, can lead to traces of different lengths. Hence, we need to model the case of variable length data. In Section 3 we present our FB algorithm for coping with a single trace. This approach extends the state space of the HMM to include a variable which counts how far one has processed along the output trace, each state transition within the HMM corresponds to the processing of a single key symbol; however, each transition may take up a varying number of output symbols. In Section 4 we describe some heuristic approaches to dealing with multiple traces, we examine the pros and

cons of each approach. In Section 5 we present some experimental results for our methods as applied to various exponentiation algorithms.

We end this introduction by thanking John Malone-Lee for various useful discussions and insight whilst the work in this paper was carried out.

2 Notation

In this section we introduce the notation we will use throughout the paper. In particular we highlight the difference between our approach and that of Karlof and Wagner.

If X is a discrete random variable then we let $p(X = x)$ denote the probability distribution function, which we shorten to $p(x)$ for compactness when the underlying random variable X is clear. We let $p(x|y)$ denote the probability that the random variable X is x, given that Y is y, a notation which is extended to $p(x_1, \ldots, x_s|y_1, \ldots, y_t)$ in the standard way.

We assume we are interested in analysing an exponentiation algorithm with respect to a given fixed, but hidden, exponent k of at most N symbols, which we shall call the key. The symbols (usually bits) of k we shall denote by the vector $\boldsymbol{k} = (k_1, \ldots, k_N)$. As the algorithm progresses the internal state of the algorithm passes through a sequence of states $\boldsymbol{q} = (q_0, \ldots, q_N)$. We assume there is one internal state transition for each key symbol. When running the exponentiation algorithm the attacker obtains a sequence of observable outputs $\boldsymbol{y} = (y_1, \ldots, y_L)$.

In the model of Karlof and Wagner the observable outputs are in one-to-one correspondence with the internal states, hence $L = N$. The values of the observables are taken from the set $\{D, AD\}$; each internal state corresponds to one of these symbols and errors in measurement are specified by given the error probability p_0 which is $p(y_n = AD|q_n = D) = p_0$ and $p(y_n = D|q_n = AD) = p_0$. The internal sequence of states $\boldsymbol{x} = (x_0, \ldots, x_N)$ of the HMM in Karlof and Wagner's approach is essentially given by $x_n = (k_n, q_n)$.

In our model the observable outputs are not in one-to-one correspondence with the internal states, hence $L \neq N$. The observables are taken from the language $\mathcal{O}$, which is generated from the alphabet $\{D, A, \emptyset, \perp\}$, where $\emptyset$ is the zero-length observable and $\perp$ denotes unknown; each internal transition corresponds to a number of these symbols. To keep track of the number of observable symbols consumed by the state transitions we introduce another variable $\boldsymbol{m} = (m_1, \ldots, m_N)$, which signals that at internal algorithm state q_n the algorithm has output a total of m_n observable symbols. In particular we have $L = m_N$. The internal state of the HMM in our approach is given by the triple $x_n = (k_n, q_n, m_n)$. The precise number of output symbols "consumed" on entering a given state depends on the previous internal state of the HMM and the new key bit. To simplify matters we assume that m_n depends only on m_{n-1} and q_n, which is the case in all algorithms under consideration.

Again we assume that internal state corresponds to one of the observables symbols in $\mathcal{O}$. Errors are then modelled by defining $p(\text{ observed} = o_j|\text{ expected} = o_i) = p_{i,j}$, for $o_i, o_j \in \mathcal{O}$,

3 HMMs with Variable Length Data

In this section we present how to use the FB algorithm to analyse our HMM for a single trace, where the length of the list of output symbols may not correspond to the length of the list of internal states. We first present the FB algorithm as a general tool, we then recap on its application to classic HMM, and finally we present the modifications needed to cope with our situation.

3.1 Forward-Backward Algorithm

The FB algorithm is an efficient method for computing all *marginal sums*

$$t_n(x_n) = \sum_{x_0} \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} f(x_0, x_1, \ldots, x_N), \tag{1}$$

for $n = 0, 1, \ldots, N$, when the function being summed has a factorisation of the form

$$f(x_0, x_1, \ldots, x_N) = \prod_{n=1}^{N} g_n(x_{n-1}, x_n). \tag{2}$$

By substituting (2) into (1) and rearranging the factors and summation signs, it is easy to see that

$$t_n(x_n) = r_n(x_n) s_n(x_n)$$

for all n and x_n, where

$$r_n(x_n) = \sum_{x_{n-1}} g_n(x_{n-1}, x_n) \sum_{x_{n-2}} g_{n-1}(x_{n-2}, x_{n-1}) \cdots$$

and

$$s_n(x_n) = \sum_{x_{n+1}} g_{n+1}(x_n, x_{n+1}) \sum_{x_{n+2}} g_{n+2}(x_{n+1}, x_{n+2}) \cdots$$

These summations can be computed recursively via

$$r_n(x_n) = \sum_{x_{n-1}} g_n(x_{n-1}, x_n) r_{n-1}(x_{n-1}) \text{ for } n = 1, 2, \ldots, N \tag{3}$$

$$s_n(x_n) = \sum_{x_{n+1}} g_{n+1}(x_n, x_{n+1}) s_{n+1}(x_{n+1}) \text{ for } n = N-1, N-2, \ldots, 0 \tag{4}$$

starting from $r_0(x_0) \equiv 1$ and $s_N(x_N) \equiv 1$.

3.2 Classic HMM

In the standard application of the FB algorithm to HMMs, $f(x_0, x_1, \ldots, x_N)$ is the joint distribution of hidden variables $(x_0, x_1, \ldots, x_N)$ and corresponding observed data $\boldsymbol{y} = (y_1, y_2, \ldots, y_N)$. The FB algorithm can then be applied since the factors of f are

$$g_1(x_0, x_1) = p(x_0)p(x_1|x_0)p(y_1|x_1) \text{ and } g_n(x_{n-1}, x_n) = p(x_n|x_{n-1})p(y_n|x_n)$$

for $n = 2, 3, \ldots$. Hence, the FB algorithm allows us to compute *marginal posteriors* since

$$p(x_n|y_1, y_2, \ldots, y_N) = t_n(x_n) / \sum_{x_n} t_n(x_n).$$

The application of Karlof and Wagner can be interpreted as taking the internal states x_n to be (k_n, q_n), where k_n is the n-th symbol of the key and q_n is the internal state. The values y_n then correspond to the observations made during the side channel analysis and the probability $p(y_n|x_n)$ models the error probability of seeing certain outputs given the internal state of the exponentiation algorithm. The formulae given to evaluate $p(x_i|\boldsymbol{y})$ given in [4] is then simply an application of the standard FB algorithm to this situation.

3.3 HMM with Variable Length Data

We now turn to the situation where the indexing of the observable states y_i does not correspond in a one-to-one manner with the indexing of the internal states x_i. The FB method really pays no regard to the indexing of the data, so that, providing the joint distribution of hidden variables and data can be factorised in the form (2), we can apply the method.

We now use a HMM in which the state variable x_n is a triple (k_n, q_n, m_n), and we assume a Markov structure, with transition probabilities

$$p(x_n|x_{n-1}) = p(k_n)p(q_n|q_{n-1}, k_n)p(m_n|m_{n-1}, q_n).$$

We assume that the distribution of the data $\boldsymbol{y}$ given $x_0, x_1, \ldots, x_n$ factorises into a product

$$p(\boldsymbol{y}|x_0, x_1, \ldots, x_n) = \prod_{n=1}^{N} d_n(\boldsymbol{y}, x_{n-1}, x_n);$$

the interpretation is that d_n is the distribution of the nth 'chunk' of data, conditional on x_{n-1} and x_n, or in practice only on (m_{n-1}, m_n, q_n). The FB algorithm can then be used, with

$$g_1(x_0, x_1) = p(q_0)p(k_1)p(q_1|q_0, k_1)p(m_1|m_0, q_1)d_1(\boldsymbol{y}, x_0, x_n)$$

(with m_0 fixed at 0) and, for $n = 2, 3, \ldots$,

$$g_n(x_{n-1}, x_n) = p(k_n)p(q_n|q_{n-1}, k_n)p(m_n|m_{n-1}, q_n)d_n(\boldsymbol{y}, m_{n-1}, m_n, q_n).$$

Note that this allows (but does not require) that the data chunk pointers m_n are random and not determined by q_n and m_{n-1}. This is a slight extra generalisation on the situation we are in when analysing exponentiation algorithms.

Given that the g_n factors, the forwards and backwards recursions (3) and (4) are performed and the marginal posteriors can be computed as above:

$$p(k_n, q_n, m_n|\boldsymbol{y}) = t_n(x_n) / \sum_{x_n} t_n(x_n)$$

from which the marginal distribution for k_n alone can be found by summing out m_n and q_n.

In the above we set $m_0 = 0$, the corresponding condition at the other end of the sequence, that restores the forwards/backwards symmetry is given as follows: If we assume that all of the observed data sequence is generated by the N steps of the HMM, then m_N is also known, and is equal to the length of $\boldsymbol{y}$. The known values of m_0 and m_N can then be regarded as part of the observed data sequence, and their values fixed by specification of the end conditions, so that $r_0(x_0) = 1$ if and only if $x_0 = (q_0, m_0)$ has $m_0 = 0$, and $s_N(x_N) = 1$ if and only if $x_N = (k_N, q_N, m_N)$ has m_N equal to the length of $\boldsymbol{y}$.

As well as specifying the initial state, q_0, as part of the HMM, we also include a set of permissible terminating states. This allows us to more accurately model algorithms with specific termination points and gives a corresponding increase in the accuracy of the calculated belief values.

3.4 Useful Properties

In addition to accommodating errors at the level of individual symbols with a given probability, our model allows the specification of an error map specifying different probabilities for each symbol transformation, i.e. the probability of reading an "A" as a "D" could be 0.5 but the probability of reading a "D" as an "A" could be only 0.1. The model also allows for symbols to be marked as unknown - so that in the case of a highly ambiguous reading it is possible to enter nothing rather than input a potentially misleading value into the HMM. Both of these types of error are handled in the d_n function during the processing of the FB algorithm.

Some algorithms may terminate without generating an output symbol for all input symbols - for example, when the remaining symbols in an exponent are all zero. We can avoid artificial pre-processing of data to fix the length of traces for such algorithms by using the zero-length observable symbol $\emptyset$, and having a corresponding terminating state in the algorithms HMM with this observable that links only to itself. This technique is used when modelling the Liardet–Smart exponentiation algorithm [5] and the Oswald–Aigner exponentiation algorithm [8].

3.5 Implementation Notes

At each n we need to store and manipulate the tables r_n and s_n indexed by $x_n = (k_n, q_n, m_n)$. While k_n and q_n have small sets of possible values, the range of values of m_n for which both r_n and s_n are non-zero varies with n. However, this does not cause a problem in practice; since each internal state change will output between $l = \min_{y \in \mathcal{Y}} |y|$ and $h = \max_{y \in \mathcal{Y}} |y|$ observable symbols, we have $ln \leq m_n \leq hn$, and also $l(N - n) \leq (m_N - m_n) \leq h(N - n)$ by symmetry. This is exploited to save time in the FB algorithm by providing an initial reduction in the portion of the state space which has the potential to generate non-zero belief values.

During the processing of the forward values, we further reduce the possible range of values for m_{n+1} at step n based of the range of values for which m_n

generated non-zero beliefs and the possible transitions from q_n. This state space reduction is also performed for m_{n-1} when calculating the backward values.

4 Modelling with Multiple Traces

Given a single trace we can use belief propagation as in Section 3 to calculate exact beliefs for each key symbol. However, trying to deal with multiple traces in this model increases the state space exponentially in the number of traces. We need a heuristic to combine the results from analysing the traces separately. This heuristic should make at most a polynomial (in number of traces and input size) number of calls to the single trace algorithm.

In the single trace version, we calculate the belief for a symbol n from a trace $\boldsymbol{y}$ and prior key symbol distribution D:

$$b_n(\boldsymbol{y}, D) = p(k_n = 1|y, D) = \sum_{q \in S} \sum_{m=0}^{|\boldsymbol{y}|} \frac{t_n^{\boldsymbol{y},D}(\{q, m, 1\})}{\sum_{x_n} t_n^{\boldsymbol{y},D}(x_n)},$$

where $t_n^{\boldsymbol{y},D}$ is the same as the t_n function in Section 3 but we now make the dependence on $\boldsymbol{y}$ and D explicit. We now present two heuristic approaches to dealing with multiple traces, one a natural analogue of that of Karlof and Wagner to the case of variable length observable data, the second one is based on *loopy belief propagation* (see, for example, [11]).

We let $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{|\mathcal{Y}|}\}$ denote the set of traces obtained from the side channel, we let N denote the number of key symbols in k and let D denote the prior probability distribution on these symbols. The goal of our heuristics is to output a heuristic posterior distribution, which we also denote by D, which takes into account the information contained within the set of traces $\mathcal{Y}$.

4.1 Karlof-Wagner

The *belief propagation* method used by Karlof and Wagner as applied to our situation is described below.

For all $\boldsymbol{y} \in \mathcal{Y}$
  For $n = 1 \ldots N$
    $D'(k_n = 1) \leftarrow b_n(\boldsymbol{y}, D)$
  $D \leftarrow D'$
Return D

In other words we compute

$$p(k_n = 1|\mathcal{Y}, D) = b_n\left(\boldsymbol{y}_1, b\left(\boldsymbol{y}_2, \ldots b\left(\boldsymbol{y}_{|Y|}, D\right)\ldots\right)\right)$$

where $b(\boldsymbol{y}, D)$ is shorthand for $\{b_n(\boldsymbol{y}, D)\}_{n=0}^{N}$.

This method has the advantage of having a complexity which is linear in the number of traces, however the final heuristic posterior distribution on the

key symbols depends heavily on the order in which the traces are processed. As an example of this problem consider the (non-randomized) right-to-left binary algorithm on a two bit exponent. Processing traces $\boldsymbol{y}_1 = ADD$ and $\boldsymbol{y}_2 = DAD$ should give $P(k_n = 1) = \frac{1}{2}$; However, this method gives the distribution $\{0.9, 0.1\}$ when processing ADD first and $\{0.1, 0.9\}$ when processing DAD first.

4.2 Bitwise Average

We investigated a number of heuristic methods for combining the data from multiple traces, although some produced results better than the following method, their complexity was too high for practical use in large examples. The following method was the one which produced the best results with a reasonable performance.

Our new heuristic combining method is as follows: We calculate the beliefs for each trace, perform a bitwise average to combine them into a new key symbol distribution and repeat until the distribution converges.

Repeat
  $D' \leftarrow D$
  For all $\boldsymbol{y} \in \mathcal{Y}$
    For $n = 1 \ldots N$
      $D_{\boldsymbol{y}}(k_n = 1) \leftarrow b_n(\boldsymbol{y}, D)$
    $D \leftarrow \mathtt{Avg}_{\boldsymbol{y} \in \mathcal{Y}}(D_{\boldsymbol{y}})$
Until $D \approx D'$
Return D

In other words we set

$$D = \left\{ \mathtt{Avg}_{\boldsymbol{y} \in \mathcal{Y}} \left(b_n \left(\boldsymbol{y}, D \right) \right) \right\}_{n=0}^{N}$$

and repeat until the values in the distribution D have appeared to converge.

Intuitively, this method is inspired by the following technique, one could think of a factor graph (see, for example, [11]) which connects the corresponding hidden states in each trace with an averaging function. As this graph contains loops, we must perform *loopy belief propagation* - that is, we start with an initial set of messages, in our case the initial key symbol distribution and the traces, and iterate until we (hopefully) get convergence. Clearly the output of this heuristic is independent of the input order of the traces, however it is unclear how many iterations are necessary and whether the method converges for a given input sequence.

The method also does not take into account information which can be obtained by considering two or more traces at once, so called *cross trace analysis*. As an example of this consider the randomized binary algorithm on a 3 bit exponent, where no errors occur when interpreting the power trace. A trace of DDD indicates that there are exactly three '0'-bits and zero '1'-bits. Given a set of traces $\{DDD, ADADAD\}$ the output should be 000 with probability one, as the

first trace shows that all of the As in the second trace are spurious. However, as there is no interaction between traces the final result is not definite.

We now turn to the discussion of the averaging function $\texttt{Avg}(\cdot)$ in the above heuristic method. If one uses the arithmetic mean then problems can arise, as the following example demonstrates: If $p(k_n = 1|\boldsymbol{y}_1, D) = 1$ and $p(k_n = 1|\boldsymbol{y}_2, D) = 0.5$ averaging these values gives us $p(k_n = 1|\boldsymbol{y}_1, \boldsymbol{y}_2, D) = 0.75$. However, $p(k_n = 1|\boldsymbol{y}_1, D) = 1$ is saying that the key symbol is *definitely* 1 whereas $p(k_n = 1|\boldsymbol{y}_2, D) = 0.5$ says that $\boldsymbol{y}_2$ gives no information about the key symbol. Clearly then we should combine the traces such that $p(k_n = 1|\boldsymbol{y}_1, \boldsymbol{y}_2, D) = 1$.

Replacing the arithmetic mean with a weighted mean where the weight function is 0 at $x = 0.5$ and increases with $|x - 0.5|$ solves this problem by allowing us to calculate the combined belief as

$$\texttt{Avg}(b_1, b_2, \ldots, b_n) = \begin{cases} 0.5 & \sum_{i=0..n} w(b_i) = 0, \\ \frac{\sum_{i=0..n} w(b_i) b_i}{\sum_{i=0..n} w(b_i)} & \text{Otherwise.} \end{cases}$$

The most effective function we have found for producing the weight is $w(x) = 4x^2 - 4x + 1$. In the above example the above weighting would give us a new combined belief of $p(k_n = 1|\boldsymbol{y}_1, \boldsymbol{y}_2, D) = 1$.

5 Performance of Heuristics

Our main interest was in analysing exponentiation algorithms in the situation where defences already exist to make it hard to distinguish doubles from additions. In such a situation one is interested in how much defence one obtains against simple power analysis by using the naive (non-randomized) binary algorithm. In addition, there are certain exponentiation algorithms which have been proposed for precisely the situation where, hopefully, indistinguishable operations have been implemented. For example in [5] Liardet and Smart propose an algorithm to be used in conjunction with their indistinguishable addition formulae so as to help mitigate against differential power analysis. They do this by introducing a small amount of randomization into the exponentiation algorithm without increasing the run time considerably.

In [10] Walter analyses the Liardet–Smart exponentiation algorithm in the situation where there are no indistinguishable operations, and from the power trace one can work out exactly the sequence of additions and doublings which are carried out. Walter shows that in such a situation, for a 160-bit exponent, one can break the Liardet–Smart algorithm with ten traces and work effort around $O(2^{64})$, using $R = 5$ in the Liardet–Smart algorithm. If one increases the number of traces to twenty then the work effort goes down to $O(2^{40})$. This result is achieved by an exact analysis of the Liardet–Smart algorithm.

With our method we were able to investigate various exponentiation algorithms with various error models. To illustrate typical results we used 160-bit exponent values and two errors models, which we now describe. Clearly other more complicated error models are allowed in our analysis but for ease of presenting our results we focus on the following two:

5.1 Error Model A:

Here we used an error model which swapped an A for a D, and vice versa, with a given fixed probability p_0,

$$p(\text{ observed } = A|\text{ expected } = D) = p(\text{ observed } = D|\text{ expected } = A) = p_0,$$
$$p(\text{ observed } = A|\text{ expected } = A) = p(\text{ observed } = D|\text{ expected } = D) = 1 - p_0.$$

5.2 Error Model B:

This model assumes that a certain fixed proportion p_0 of the symbols are unable to be read.

$$p(\text{ observed } = \perp \mid \text{expected } = D) = p(\text{ observed } = \perp \mid \text{expected } = A) = p_0.$$
$$p(\text{ observed } = A|\text{ expected } = A) = p(\text{ observed } = D|\text{ expected } = D) = 1 - p_0.$$

For each algorithm we performed a number of experiments, and produced the results in Tables 1, 2 and 3, for the standard binary algorithm, the Liardet–Smart algorithm and the Oswald–Aigner algorithm (OA2). In these tables the column $\mathfrak{p}$ represents the average proportion of key bits correctly recovered. Given this we can compute the amount of additional work needed to recover the key, given that the HMM algorithm recovers the stated proportion of the key symbols. This workfactor is derived using the low-Hamming weight variant of the Baby-Step/Giant-Step algorithm for the discrete logarithm problem (DLP) [9]. Namely, if we can derive an approximation to a discrete logarithm such that proportion $\mathfrak{p}$ of the N bits are correct, then one can solve for the exact discrete logarithm in expected time

$$O\left(\sqrt{N \cdot \mathfrak{p}} \cdot \binom{N/2}{N \cdot \mathfrak{p}/2}\right),$$

which may be more efficient than the $O(2^{N/2})$ technique of using the standard Baby-Step/Giant-Step algorithm. For 160-bit exponents we obtain an improvement as soon as $\mathfrak{p} > 0.8$.

Table 1. Results for the Binary Exponentiation Algorithm

Error Model	p_0	Number of Traces 1 $\mathfrak{p}$	5 $\mathfrak{p}$	10 $\mathfrak{p}$	20 $\mathfrak{p}$	100 $\mathfrak{p}$
–	0.0	1.00	1.00	1.00	1.00	1.00
A	0.1	0.66	0.75	0.77	0.79	0.82
	0.2	0.59	0.67	0.68	0.69	0.70
B	0.1	0.80	0.87	0.89	0.89	0.90
	0.2	0.72	0.77	0.77	0.77	0.78

Table 2. Results for the Liardet–Smart Exponentiation Algorithm ($R = 5$)

Error Model	p_0	Number of Traces 1 $\mathfrak{p}$	5 $\mathfrak{p}$	10 $\mathfrak{p}$	20 $\mathfrak{p}$	100 $\mathfrak{p}$
–	0.0	0.40	0.62	0.79	0.89	0.97
A	0.1	0.43	0.50	0.55	0.59	0.62
	0.2	0.39	0.44	0.47	0.49	0.52
B	0.1	0.43	0.51	0.57	0.63	0.71
	0.2	0.39	0.45	0.48	0.51	0.57

Table 3. Results for the OA2 Exponentiation Algorithm

Error Model	p_0	Number of Traces 1 $\mathfrak{p}$	5 $\mathfrak{p}$	10 $\mathfrak{p}$	20 $\mathfrak{p}$	100 $\mathfrak{p}$
–	0.0	0.82	0.89	0.95	0.97	0.98
A	0.1	0.59	0.69	0.72	0.77	0.79
	0.2	0.54	0.63	0.67	0.70	0.73
B	0.1	0.66	0.79	0.82	0.83	0.84
	0.2	0.60	0.71	0.72	0.75	0.76

To compare our heuristic trace combining method to that used by Karlof and Wagner we present in Table 4 the average proportion of key bits recovered correctly using our error models, but using the heuristic belief propagation method of Karlof and Wagner.

From the tables we see that our trace combining method recovers more of the key than the trace combining method of Karlof–Wagner. Furthermore, the results in Table 4 for both Liardet–Smart and Oswald–Aigner without errors show a decrease in accuracy as the number of traces increases. This is due to the way their heuristic overemphasises the belief values generated by early traces and causes the belief values to increase very rapidly when given consistent trace data, but not decrease as rapidly when given evidence to the contrary. With the Liardet–Smart algorithm an A in the trace indicates that either the current or previous key bit is 1 whereas a D indicates that the current key bit is marginally more likely to be 0 than 1. When processing multiple traces, the Karlof-Wagner heuristic combines these slight biases until rounding errors in the floating point representation cause a belief of 1 in the key bit being a 0. If a future trace then indicates that the key bit is actually 1 a contradiction occurs in the forward–backward algorithm causing it to fail. A non-zero probability of error prevents the belief values from getting close enough to 1 for such a rounding error to occur, which explains why the heuristic does not fail in those case. The decrease in accuracy when processing the Oswald-Aigner algorithm is due to the same reason; a slight bias for a particular key bit value is compounded until it outweighs strong evidence to the contrary.

Table 4. Results for the Karlof–Wagner Heuristic

Error Model	p_0	Algorithm: Binary 1	Binary 20	Binary 100	L–S 1	L–S 20	L–S 100	OA2 1	OA2 20	OA2 100
–	0.0	1.0	1.0	1.0	0.42	0.00	0.00	0.82	0.39	0.39
A	0.1	0.66	0.72	0.72	0.29	0.48	0.53	0.64	0.76	0.77
	0.2	0.61	0.68	0.68	0.21	0.42	0.48	0.58	0.69	0.73
B	0.1	0.76	0.85	0.85	0.27	0.53	0.54	0.66	0.77	0.78
	0.2	0.67	0.78	0.78	0.17	0.47	0.50	0.62	0.74	0.75

We also see that Error Model B allows one to recover more of the key than Error Model A, this is because in Error Model A we feed the algorithm possibly incorrect information, whilst in Error Model B we only tell it when we are unsure of certain data values; a value of $\perp$ provides only correct information, whereas an observation error provides misinformation. The ability to mark that an observable action took place without having to commit to what the action was would be very useful in practice, where two sub-operations may have similar power-traces or timing characteristics.

We see that, in the case of the Liardet–Smart algorithm, our general HMM based method is almost as effective as Walter's method in the case of no errors and multiple traces. However, the results for a single trace with no errors appear to be worse than random guessing! This doesn't quite tell the whole story, as the 40% (on average) of correct bits have a high level of confidence in correctness whilst the remaining bits have very low level. Using this level of confidence to perform a weighted average when combining multiple traces is key to our trace combining heuristic.

This is the first work to look at the Liardet–Smart algorithm in the case of noisy trace data, as would happen when the algorithm is used in the context of indistinguishable addition/doubling formulae as proposed in [5]. In this situation the Liardet–Smart algorithm is relatively immune to simple power analysis via our HMM method. This contrasts with the case of the Oswald–Aigner method, which performs only marginally better than the standard binary method if there are multiple traces with errors.

References

1. I.F. Blake, G. Seroussi and N.P. Smart, editors. *Advances in Elliptic Curve Cryptography.* Cambridge University Press, 2005.
2. É. Brier, I. Déchène and M. Joye. Unified addition formulæ for elliptic curve cryptosystems. In *Embedded Cryptographic Hardware: Methodologies and Architectures.* Nova Science Publishers, 2004.
3. M. Joye and J.-J. Quisquarter. Hessian elliptic curves and side-channel analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, Springer-Verlag LNCS 2162, 402–410, 2001.

4. C. Karlof and D. Wagner. Hidden Markov model cryptanalysis. In *Cryptographic Hardware and Embedded Systems – CHES 2003*, Springer-Verlag LNCS 2779, 17–34, 2003.
5. P.-Y. Liardet and N.P. Smart. Preventing SPA/DPA in ECC systems using the Jacobi form. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, Springer-Verlag LNCS 2162, 391–401, 2001.
6. E. Oswald. Enhancing simple power-analysis attacks on elliptic curve cryptosystems. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, Springer-Verlag LNCS 2523, 82–97, 2002.
7. E. Oswald. Side-Channel Analysis. In [1], 69–86, 2005.
8. E. Oswald and M. Aigner. Randomized addition-subtraction chains as a countermeasure against power attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, Springer-Verlag LNCS 2162, 39–50, 2001.
9. D. Stinson. Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. *Math. Comp.*, **71**, 379–391, 2002.
10. C. Walter. Breaking the Liardet–Smart randomized exponentiation algorithm. In *Proceedings Cardis '02*, 59–68, USENIX Assoc., 2002.
11. J.S. Yididia, W.T. Freeman and Y. Weiss. Understanding Belief Propagation and its Generalizations. Mitsubishi Electric Research Laboratories Technical Report TR-2001-22, January 2002.