Recovering Secret Keys from Weak Side Channel Traces of Differing Lengths

Colin D. Walter

Comodo CA Research Laboratory
7 Campus Road, Bradford, BD7 1HR, UK
Colin.Walter@comodo.com

Abstract. Secret key recovery from weak side channel leakage is always a challenge in the presence of standard counter-measures. The use of randomised exponent recodings in RSA or ECC means that, over multiple re-uses of a key, operations which correspond to a given key bit are not aligned in the traces. This enhances the difficulties because traces cannot be averaged to improve the signal-to-noise ratio.
The situation can be described using a hidden Markov model (HMM) but the standard solution is computationally infeasible when many traces have to be processed. Previous work has not provided a satisfactory way out. Here, instead of *ad hoc* sequential processing of complete traces, trace prefixes are combined naturally in parallel. This results in the systematic extraction of a much higher proportion of the information theoretic content of the leakage, enabling many keys of typical ECC length to be recovered with a computationally feasible search through a list of most likely values. Moreover, likely errors can now be located very easily.

Key Words. Side channel leakage, simple power analysis, SPA, Hidden Markov Models, Forward-Backward Algorithm, Viterbi Algorithm.

1 Introduction

Side channel leakage from embedded cryptographic devices may contain substantial information. When possible this is averaged to improve the signal to noise ratio and enable recovery of the secret key. However, some randomised exponentiation algorithms are designed so that averaging over contemporaneous operations reduces rather than increases the useful information [4, 8, 9, 15]. With a perfect side channel that distinguishes squares from multiplications in each trace, it is possible to recover the secret exponent key for most of these algorithms without substantial effort [12, 13, 15].

In the real world the side channels are rarely so clear, especially where designers have taken steps to reduce the leakage. Then standard statistical techniques can be applied to detect correlations between possible key bits and the trace data, but they are computationally infeasible for the length of cryptographic keys and the expected level of leakage. Instead, dedicated algorithms are required to extract meaningful information and search for the most likely keys.

Karlov and Wagner [5] modelled this using input-driven Hidden Markov Models, and suggested the sequential processing of complete traces as an effective way of limiting the computational complexity. However, they only consider traces of equal length where the ith observation always corresponds to the ith exponent digit. Green, Noad and Smart [3] show how to deal with the traces of different lengths which are more typical of randomised exponentiation algorithms, and they provide some heuristic methods for the sequential processing of complete traces. Nevertheless, even with strong leakage it is clear from their tables (e.g. *op. cit.* Table 1) that very little of the information content of the traces is successfully extracted. Moreover, it is unclear whether their methods would converge to any solution in the presence of weak leakage.

The hidden Markov model (HMM) of [5] and [3] leads to a forward algorithm that provides a *global* minimum for a metric that measures distance from a best-fit solution. A backward algorithm then generates the state sequence and hence the input key which yields that minimum value. On the other hand, the randomised exponentiation algorithms of interest perform recodings which influence the side channel traces only *locally* for a small number of operations. Therefore it seems better to attempt recovery of input key bits using a more locally-based algorithm. This, of course, is an approach which proved very successful with weak side channels in the original timing and power analysis attacks of Kocher *et al.* [6, 7]. Their averaging of contemporaneous trace outputs does not work here because such values no longer correspond to the same input symbol. Thus, some new ideas are required but, to benefit from very weak leakage, averaging is still key to avoiding previous convergence problems of [5] and [3].

Here the proposed algorithm adjusts trace positions in an attempt to align outputs which correspond to the same input symbol. This allows averaging to take place and also makes it possible to take into account the influence of recoding decisions on neighbouring operations. There are a number of parameters to choose in the algorithm. Their choice enables the calculations to be kept within available computational resources. The output is a set of good guesses at the secret key. Moreover, each bit is naturally assigned a correctness probability which enables likely errors to be located easily. This seems to be a new feature: Green *et al.* [3] do not say if they can locate possible errors, but the ability enables many more errors to be corrected. Simulation results are provided to show what fraction of keys are recoverable for a given effort.

As the whole process is computationally feasible, the immediate conclusion is that designers should assume most leakage can be converted successfully into useful knowledge when a secret key is re-used with these random recoding exponentiation algorithms. Indeed, they should be able to calculate upper bounds on the number of times the key can be safely re-used.

2 The Leakage Model

The context of the attack is the repeated use of a randomised exponentiation algorithm for computing M^D in any cryptographic group where D is a fixed

secret key which is not subject to blinding by a random multiple of the group order, and M is an unknown ciphertext which may vary and may be whitened. The adversary is assumed to know all the details of the exponentiation algorithm. Use of the key provides him with a side channel trace for the exponentiation itself, but no further information is assumed: he is not expected to be able to choose any input, view any output, or usefully observe any pre- and post- processing.

It is assumed that occurrences of multiplicative operations in the exponentiation can be identified accurately from the corresponding side channel trace, but that their identities as squares or multiplications can only be determined with a substantial degree of inaccuracy[1] [2]. The adversary's aim is to discover D using computationally feasible resources.

3 The Randomised Exponentiation

Examples of the randomised exponentiation algorithms which can be attacked in the way described here include those of Liardet-Smart [8], Oswald-Aigner [9] and Ha-Moon [4, 15]. Their common, underlying basis is a recoding of the binary representation of the key D into a form

$$D = ((...(d_{m-1}2^{m_{m-2}} + d_{m-2})2^{m_{m-3}} + ... + d_2)2^{m_1} + d_1)2^{m_0} + d_0$$

where the digits $d_i \in \mathcal{D}$ and 2-power exponents $m_i \in \mathcal{M}$ belong to some fixed, pre-determined sets $\mathcal{D}$ and $\mathcal{M}$ respectively. Both d_i and m_i are selected according to some finite automaton which has the bits of D and the output from a random number generator (RNG) as inputs. Different bit streams from the RNG result in different recodings of D.

The exponentiation M^D begins with the pre-calculation of the table $\{M^d \mid d \in \mathcal{D}\}$. Then the main iterative step of the exponentiation consists of m_i squarings followed by a multiplication by the table value M^{d_i} when $d_i \neq 0$. This results in a sequence of multiplicative operations which is most easily presented using d_i to denote multiplication by the table entry M^{d_i} and m_i copies of 0 to denote the squarings. We call this a *recoding sequence* for D. For example, the exponent $D = 13_{10} = 1101_2$ has a recoding $D = (1.2^2+3)2^1+\bar{1}$ which gives the operation sequence $100301\bar{1}$. (Alternatives in processing the first digit are ignored.)

The exponentiation algorithms of interest here are assumed to have the property that perfect knowledge of the multiplication/squaring sequences for a small number of recodings of D yields enough information to reconstruct the secret key D with at most a small number of ambiguities. This is the case for the algorithms mentioned above: attacks on them using such information are described in [12], [13] and [15] respectively.

[1] For the table-driven exponentiation algorithms under attack here it may also be possible, with a degree of uncertainty, to identify which table element is used in a multiplication, and hence guess at the most likely value for the exponent digit. The methods below can be extended easily to such cases.

4 Notation for Leakage Traces

The above-mentioned exponentiation algorithms are reasonably secure when the key D is used only once, as in ECDSA [1], because a single recoding pattern does not generally yield sufficient information to determine a computationally feasible search space for the key. However, for key re-use, the pattern must be hidden. Hence implementers generally employ both hardware and software counter-measures to prevent leakage of the recoding pattern through any side channels. Consequently, an attacker only obtains partial information about any recoding through side channels. But extensive testing of the cryptographic system and some pre-computation enables the adversary to process the side channel information from each exponentiation into a sequence of probabilities that each component operation is a squaring. For convenience, this will be referred to as a *trace*. So the operation sequence $10030\bar{1}$ yields a trace such as (0.23, 0.87, 0.69, 0.15, 0.83, 0.42), with the larger probabilities occurring for doublings.

For consistency, exponent bit strings, recoding sequences and traces are all written in the same left-to-right order. For convenience, this is the order in which the recoding process consumes key bits and generates operation sequences. Then, in an obvious sense, a prefix in one list always corresponds to some prefix in another. These prefixes are extended incrementally as the attack progresses.

Recodings are pairs consisting of (i) the operation sequence r which the recoding automaton has generated for D, and (ii) the state s which the automaton has entered at that point. $\mathcal{R}(D)$ denotes the set of all these recodings (r, s) of D. Let $\bullet$ denote the end-of-list symbol and also the final state of the recoding automaton. A list is called *terminated* or *un-terminated* according to whether or not it ends with this symbol. When the recoding automaton reads $\bullet$ at the end of D, it performs the post-processing stage required to reach its final state $\bullet$ and then stops. Recoding $D\bullet$ will result in a pair $(r\bullet, \bullet) \in \mathcal{R}(D\bullet)$ for which r applied to M yields M^D. r is terminated with $\bullet$ to indicate that the operation sequence is complete. If D is not terminated, the pair $(r, s) \in \mathcal{R}(D)$ has $s \neq \bullet$ and it can be extended to a recoding of any D' with prefix D.

In the above example with $D = 13_{10}$, both $r = 10030\bar{1}$ with borrow 0 and $r' = 10030$ with borrow 1 could represent the output and state of the recoding automaton after processing D when further input bits are possible. They belong to $\mathcal{R}(D)$. Reading $\bullet$ next gives $(r\bullet, \bullet) \in \mathcal{R}(D\bullet)$ from $(r, 0)$ as no further processing is required. However, from $(r', \bar{1})$ there needs to be post-processing to obtain an element in $\mathcal{R}(D\bullet)$, for example, by appending $\bar{1}$ to obtain $(r'\bar{1}\bullet, \bullet)$.

5 The Metric

A metric $\mu(D, T)$ is constructed to provide a measurement of how well a bit string D matches the side channel leakage presented in a set T of traces. Roughly speaking, the "best" guesses at the secret key D are those strings which provide the smallest distances under this metric. As indicated above, the metric for a

set of traces T is just the average of the value of the metric for a single trace t:

$$\mu(D,T) = |T|^{-1} \sum_{t \in T} \mu(D,t)$$

Here $\mu(D,t)$ is the minimum of the metric applied to t and a single recoding $r \in \mathcal{R}(D)$, i.e.

$$\mu(D,t) = \mathrm{Min}\{\mu(r,t) \mid r \in \mathcal{R}(D)\}$$

If r is an un-terminated recoding which is no longer than trace t, i.e. $len(r) \leq len(t)$, then we define

$$\mu(r,t) = len(r)^{-1} \sum_{0 \leq i < len(r)} (1 - t_i(r_i))$$

where $r = (r_0, r_1, ..., r_{len(r)-1})$, $t = (t_0, t_1, t_2, ...)$, and $t_i(r_i)$ is the probability[2] observed through the side channel that the ith element of trace t corresponds to the same operation as r_i. The same definition is also used for $\mu(r,t)$ if r is a terminated recoding such that $len(r) = len(t)$. If $len(r) > len(t)$ then t is too short to correspond to the recoding r and so we define $\mu(r,t) = \infty$ whether r is terminated or not. Similarly, if r is a terminated recoding with $len(r) < len(t)$ then r is too short to correspond to trace t and again we set $\mu(r,t) = \infty$.

Scaling by $len(r)$ prevents shorter recodings of D being given an unjustified selection bias. Then, being the average of a set of probabilities in most cases, $\mu(r,t)$ lies in $[0,1] \cup \infty$. Hence $\mu(D,t) \in [0,1] \cup \infty$ and $\mu(D,T) \in [0,1] \cup \infty$. Clearly $\mu(r,t)$ is small when the operations in r are those which have high probability in the corresponding initial segment of t. Thus $\mu(D,T)$ is small when an initial segment of each trace in T closely matches some recoding of D.

If trace t is too short or too long to correspond to any recoding of D, then the above definitions give $\mu(D,t) = \infty$, and therefore $\mu(D,T) = \infty$ for any T containing t. Suppose d_{max} is the largest digit in $\mathcal{D}$ and k the length of the shortest trace in T. Then the shortest trace represents the leakage from too few operations to correspond a recoding of any string satisfying $D > d_{max} 2^k$. Hence $\mu(D,T)$ will only be finite for D with at most $k + \lfloor \log_2 d_{max} \rfloor$ bits.

The main problem with evaluating μ is that for cryptographically sized keys D, $\mathcal{R}(D)$ is too large a set over which to compute a minimum – it is exponential in the bit length of D. Consequently, we use an approximation

$$\mu'(D,t) = \mathrm{Min}\{\mu(r,t) \mid r \in \mathcal{S}_t(D)\}$$

to $\mu(D,t)$ which is determined iteratively by the best attempts to minimise μ for shorter strings. Specifically, for each trace and a suitable parameter R, we iteratively create a set $\mathcal{S}_t$ of up to R triples (d,i,s) which consist of the metric value $d = \mu(r,t)$ for an underlying "good" recoding r of D, the number i of

[2] It may be desirable to augment this definition of $t_i(r_i)$ to take account of the relevant transition probability from the recoding FA, so that less likely events would yield larger contributions to the sum for $\mu(r,t)$.

operations in r, and the state s of the recoding finite automaton after generating r from D. Set $\mathcal{S}_t$ is constructed as follows. First, for D being the empty bit string, $\mathcal{S}_t$ is initialised to contain just the triple $(0, 0, s_0)$ corresponding to the empty recoding sequence when the recoding automaton is initialised with start state s_0. For the iterative step, suppose $D = D'b$ for bit b, and $\mathcal{S}_t{}'$ is the set of triples constructed for D'. Then, for each $(d', i', s') \in \mathcal{S}_t{}'$, s' and b are fed into the FA. The output is a set of new states s and operations to extend the underlying recoding of D' to ones for D. These are used to create new triples (d, i, s) for D where i comes from increasing i' by the number of these operations, and d comes from a scaled incrementing of d' by the terms $(1 - t_{i''}(r_{i''}))$ with $i' \leq i'' < i$. As the triple also depends on a random input to the FA, there can be several new triples for each one in $\mathcal{S}_t{}'$. These are then rationalised by removing triples (d_2, i, s) for which there is already a triple (d_1, i, s) with $d_1 \leq d_2$. Then the R triples with the smallest values for d are chosen for inclusion in $\mathcal{S}_t$.

If the recoding automaton had s possible states and t_{max} were the maximum length of any trace, then st_{max} would be an upper bound on the size of $\mathcal{S}_t$. Hence

$$\mu'(D, T) = \mu(D, T) \quad \text{for } R = st_{max}$$

since all the smallest intermediate values for μ are retained. Whatever R is, μ' can be computed easily and accurately in time which is polynomial rather than exponential in $\log D$. It avoids enumerating all the recodings of D. R can often be picked much smaller than st_{max} without significantly affecting the accuracy of the method, and this helps reduce the complexity of the attack.

6 The Search Tree

The main phase of the attack is the construction and pruning of a (nearly) binary tree where internal edges are labelled by bits and edges to leaves by the end-of-list symbol $\bullet$. Each node N is labelled with the (possibly terminated) bit string D_N given by concatenating the labels along the branch from the root to N. Nodes are also labelled with $\mu_N = \mu'(D_N, T)$ and the sets of triples $\mathcal{S}_{t,N}$ $(t \in T)$ for $D = D_N$, each of which is computed incrementally as described in §5. The root ρ is labelled by the empty string $D_\rho = \epsilon$, $\mu_\rho = \mu(\epsilon, T) = 0$ and $\mathcal{S}_{t,\rho} = \{(0, 0, s_0)\}$. For each non-terminated node N, up to three child nodes are constructed with edges labelled 0, 1 and $\bullet$ respectively. Only the first two can grow further branches, so the tree is almost binary. Upon completion, a set of possible keys D is obtained from the labels D_N on the leaves N, and they can be arranged in order of likelihood using the values μ_N.

The tree is constructed breadth-first to aid pruning. Pruning is driven by the values of μ_N, although not quite directly. There are three pruning rules which are applied whenever possible. First, nodes N with $\mu_N = \infty$ are deleted because D_N cannot be the correct key, nor a prefix of the correct key. This limits the depth of the tree, thereby ensuring the construction terminates. It also removes leaves near the root so that leaves only appear towards the bottom of the tree.

Secondly, un-terminated nodes whose children have all been pruned are also deleted because every line of descent from them eventually leads to a problem.

The third, and final, pruning rule uses two threshold parameters. The first, B, is the maximum breadth of the tree after pruning. The second, λ, is the number of "lookahead" bits. These parameters are chosen to make the construction of the whole tree and subsequent calculations computationally feasible.

The *level* of a node N is its distance from the root, namely $len(D_N)$. Suppose the tree has been fully constructed down to level $l+\lambda$ and all pruning rules have been applied to the levels above l. For a node N at level l, let $\mathcal{N}_N$ be the set of all its un-terminated descendents at level $l+\lambda$, and its terminated descendents with level at most $l+\lambda$. Let[3] $\bar{\mu}_N = \min\{\mu_{N'} \mid N' \in \mathcal{N}_N\}$. Then the set of nodes at level l is pruned to leave the B nodes with the smallest values $\bar{\mu}_N$.

This rule removes the nodes whose recodings provide the poorest match to the observed leakage. Since recoding choices affect the pattern of subsequent operations and this effect may only become apparent in the metric after processing several more bits, larger values of λ tend to give better results in determining the best match D. Larger values of B clearly make the inclusion of the correct key D more likely. Even the best fit key with smallest value at its leaf node may not have the smallest value for μ_N or $\bar{\mu}_N$ at each intermediate node N. So B and λ must be kept large enough to include the correct key; their values can be determined only after practical experiment on the leaking device. It also pays to be increasingly light handed in pruning the final λ levels.

7 Locating Bit Errors

By their nature, all key searching algorithms suffer from unavoidable deficiencies: one is that the best fit key may not be among the good keys which they generate; another is that the correct key may not be the best fit. The first problem arises because the ultimately best fit key is not always the best at intermediate points. To ameliorate this, a number of the best keys need to be continued all the time. This action should also solve the second problem and is achieved by appropriate choice of the parameters, as illustrated in the tables of the next section.

However, there are more subtle causes of errors. A single bit error can completely de-rail the process for several reasons. Firstly, because of nature of the exponentiation algorithm, it may be possible for different key bit sequences to generate identical leakage. An example is described in Appendix 2, and it is indeed a main cause of errors when dealing with the Ha-Moon recoding algorithm. Secondly, there may be so many good choices when the wrong bit is chosen that the correct one does not survive.

Most of these bit errors are at predictable points, namely those for which the relative difference in the values of $\bar{\mu}$ for the 0-bit and 1-bit choices is very small. Specifically, the set $\mathcal{N}_N$ is partitioned into subsets $\mathcal{N}_N^{(0)}$ and $\mathcal{N}_N^{(1)}$ corresponding

[3] Different definitions of $\bar{\mu}$ are possible: e.g. one might weight the metric contribution at level $l+i$ by i^{-1} instead of 1 for $i > 0$.

to the 0- and 1- branches at N and the minimum metric values are compared for the two sets, where the sums only include terms $1-t_i(r_i)$ from levels $l+1$ to $l+\lambda$. The difference is a useful measure of confidence in the decision. It enables the search for the correct key to be prioritised by trying alternatives for the most doubtful bit decisions first. This reduces the search cost very considerably.

Finally we note that the smaller μ is at the end of the process, the better the fit, and so, on average, the fewer the number of errors. Hence it is possible to select the most likely candidates to break. Sometimes the traces can become incorrectly aligned during key recovery. This leads to a large number of errors and a high value for the metric, but such cases are easily detected and avoided.

8 Example Simulation

This section contains results from a simulation of the attack applied to the Ha-Moon algorithm [4], showing variations from the choice of parameters. The metric difference described in the previous section was used to order the bit decisions for 192-bit keys, and the bit error with the highest difference between the choices 0 or 1 was located. The probability of it lying at a particular point in this list was recorded. The tables show that errors are strongly associated with smaller differences. Consequently, for example, from Table 1 there is a probability of 0.3868+0.0058 that, with the stated parameters and 0.4 leakage, all the bit errors will be among the $\frac{2}{16}$th of bits with the lowest difference. In practice, this means checking the alternatives for only 24 bits in order to have a good probability of recovering a key. This is clearly computationally feasible. The last column states that on average only 9.082 of these bits will be in error, and so $\frac{1}{2}\binom{24}{9} \approx 2^{19}$ key tests is a realistic average for the effort involved. For the tabulated cases, the total number of errors is typically around 20 if the position of the worst error is anywhere in the top half of the ordered bit list, but, with extra computation, it can become under 11, as happens for the last line of Table 3, which covers 99.5% of all cases.

The investigation did not assess performance on the final λ bits, but any variation in the recovery rate of those bits would not affect the computational complexity significantly.

For the simulation, the trace probability values were approximately normally distributed with expectation $\frac{1}{2}(E+1)$ where E is the "strength of leakage" value in the first column of Table 1[4]. A leakage of 0 means probability $\frac{1}{2}$ that the operation is a squaring rather than a multiplication, i.e. no information content.

Tables 1 and 2 illustrate the effect of different amounts of leaked data on the key recovery process. Given that the work involved is proportional to the number of traces $|T|$ and to the number of recoding choices R which are stored, but exponential in the lookahead distance λ, it is clear from Tables 3 to 5 that the most efficient way of recovering the highest number of Ha-Moon recoded exponents is by increasing R.

[4] Green *et al.* [3] use a simpler model: trace probabilities are 0 or 1, with average $\frac{1}{2}(E+1)$. They need stronger leakage, and tabulate only $E = 0.6$ and $E = 0.8$.

Leakage Level	In 1st Half	In 3rd Quarter	In 7th Eighth	In 15th Sixteenth	In last Sixteenth	#Bit errors when all in last Eighth
0.10	1.0000	0.0000	0.0000	0.0000	0.0000	—
0.20	1.0000	0.0000	0.0000	0.0000	0.0000	—
0.30	0.9697	0.0270	0.0031	0.0002	0.0000	10.00
0.35	0.5760	0.2212	0.1542	0.0478	0.0008	9.414
0.40	0.1186	0.1456	0.3433	0.3868	0.0058	9.082
0.45	0.0143	0.0358	0.1956	0.7407	0.0136	8.925
0.50	0.0019	0.0081	0.0811	0.8913	0.0176	8.853
0.60	0.0000	0.0001	0.0304	0.9500	0.0195	8.790

Table 1. Distributions for the Worst Error in a 192-bit best-fit Exponent as Leakage varies for Ha-Moon Exponentiation [4] with $|T| = 10$, $\lambda = 5$, $R = 10$, $B = 2$.

Traces $\|T\|$	In 1st Half	In 3rd Quarter	In 7th Eighth	In 15th Sixteenth	In last Sixteenth	#Bit errors when all in last Eighth
1	0.9994	0.0006	0.0000	0.0000	0.0000	—
2	0.9344	0.0656	0.0000	0.0000	0.0000	—
3	0.7140	0.2218	0.0642	0.0000	0.0000	—
4	0.5154	0.1986	0.2860	0.0000	0.0000	—
5	0.3873	0.2010	0.4090	0.0027	0.0000	12.07
6	0.3081	0.1984	0.4579	0.0356	0.0000	11.08
8	0.1923	0.1830	0.3849	0.2394	0.0004	10.17
10	0.1186	0.1456	0.3433	0.3868	0.0058	9.082
20	0.0095	0.0369	0.1850	0.4619	0.3067	5.847
40	0.0000	0.0025	0.0403	0.2456	0.7116	3.754

Table 2. Distributions for the Worst Error in a 192-bit best-fit Exponent as the Number of Traces varies for Ha-Moon Exponentiation with 0.4 leakage, λ=5, R=10, B=2.

In the case of Liardet-Smart recoding [8], it is much harder to extract the correct key than for the Ha-Moon recoding because it is much more difficult to align the traces correctly. For example, for a maximum base 2^4, digits 0, ±1, ±3, ±5, ±7, 0.4 leakage, $\lambda = 5$ and $B = 2$, but taking 30 traces and $R = 30$, 0.38 of 192-bit exponents have all errors located in the last eighth of the ordered bit list. For 0.38 of cases the worst error occurs in the first half of the list, but, on average key guesses have fewer than 7.5 bit errors, so it is still computationally feasible to recover almost all keys.

The Oswald-Aigner recoding [9] has comparable strength to that of Ha-Moon: with the reference values of 0.4 leakage, 10 traces, λ=5, R=10 and B=2, 0.2714 of 192-bit exponents have all errors located in the last eighth of the ordered bits.

The standard, deterministic binary method is also susceptible to the algorithm. Coding decisions are unique and do not propagate to other positions, so only λ=1, R=1 and B=1 make sense. With 0.3 leakage and 10 traces, over 98% of 192-bit exponents are recovered with no errors at all. Unlike that of Green *et*

Recodings R	In 1st Half	In 3rd Quarter	In 7th Eighth	In 15th Sixteenth	In last Sixteenth	#Bit errors when all in last Eighth
6	0.5547	0.2508	0.1599	0.0341	0.0005	9.431
7	0.3576	0.2871	0.2668	0.0864	0.0021	9.023
8	0.2541	0.2359	0.3268	0.1807	0.0025	9.264
9	0.1662	0.1929	0.3584	0.2777	0.0048	9.029
10	0.1186	0.1456	0.3433	0.3868	0.0058	9.082
12	0.0605	0.0876	0.2930	0.5516	0.0073	9.012
15	0.0228	0.0472	0.2208	0.6999	0.0093	8.945
20	0.0104	0.0175	0.1553	0.8045	0.0123	8.960
30	0.0051	0.0112	0.1087	0.8579	0.0171	8.755

Table 3. Distributions for the Worst Error in a 192-bit best-fit Exponent as the Number of Recodings varies for Ha-Moon Exponentiation with 0.4 leakage, $|T|$=10, λ=5, B=2.

Lookahead λ	In 1st Half	In 3rd Quarter	In 7th Eighth	In 15th Sixteenth	In last Sixteenth	#Bit errors when all in last Eighth
1	0.2456	0.2037	0.3370	0.2107	0.0030	9.331
2	0.2019	0.1873	0.3277	0.2782	0.0049	9.222
3	0.1632	0.1685	0.3403	0.3220	0.0060	9.269
4	0.1378	0.1572	0.3341	0.3659	0.0050	9.209
5	0.1186	0.1456	0.3433	0.3868	0.0058	9.082
6	0.0997	0.1348	0.3502	0.4095	0.0058	9.039
8	0.0829	0.1380	0.3423	0.4312	0.0056	8.977

Table 4. Distributions for the Worst Error in a 192-bit best-fit Exponent as the Lookahead Value λ varies for Ha-Moon Exponentiation with 0.4 leakage, $|T|$=10, R=10, B=2.

al. [3], this algorithm reduces to the obvious, and probably optimal, one for the binary algorithm: it simply averages the leakage from each operation to see if a squaring is more or less likely than a multiplication followed by a squaring.

9 Complexity

For algorithm complexity, constant time and space is assumed for individual machine-level instructions, i.e. they are independent of the volume of data and the required arithmetic accuracy. It is also assumed that generation and storage of all possible recodings of a single input digit require $O(1)$ time and space.

There are two main terms in the time complexity for processing level l of the search tree. First, tree construction consists primarily of incrementing the metric values at level $l+\lambda$ to those for level $l+\lambda+1$. This takes $O(2^\lambda BTR \log R)$ time since there are $O(2^\lambda B)$ nodes to consider, each having T traces with R recodings apiece. Each recoding is extended in all possible ways – constant time order – but it takes $O(R \log R)$ time to select the R best recodings to keep. Secondly, pruning

Tree Width B	In 1st Half	In 3rd Quarter	In 7th Eighth	In 15th Sixteenth	In last Sixteenth	#Bit errors when all in last Eighth
1	0.1521	0.1534	0.3285	0.3613	0.0047	9.199
2	0.1186	0.1456	0.3433	0.3868	0.0058	9.082
4	0.0983	0.1493	0.3348	0.4103	0.0073	9.038
8	0.0882	0.1313	0.3286	0.4440	0.0079	9.012
12	0.0857	0.1325	0.3346	0.4378	0.0094	8.926
16	0.0898	0.1357	0.3233	0.4433	0.0079	8.949

Table 5. Distributions for the Worst Error in a 192-bit best-fit Exponent as the Tree Width B varies for Ha-Moon Exponentiation with 0.4 leakage, λ=5, $|T|$=10, R=10.

is dominated by the $O(B \log B)$ time required to order nodes. The first of these terms is the most likely to dominate for expected choices of the parameters. Both must also be multiplied by the bit length of the key, *viz.* $\log D$, to obtain the time for processing complete traces. The space complexity has two contributions. The first is $O(2^\lambda BRT)$ for storing details of the R recodings per trace associated with the $O(2^\lambda B)$ nodes between levels l and $l+\lambda$ during tree construction. The other is $O(B \log D)$ for storing details of nodes in the completed, pruned part.

10 Conclusion

A computationally feasible algorithm has been presented for determining the secret key used repeatedly in exponentiations where there is weak side channel leakage and randomised recoding has been employed to nullify the leakage. It has been shown that it is still frequently possible to recover the key. Moreover, it is easy to determine which results have few bit errors, and it is easy to locate the potential errors.

Acknowledgement

The author would like to thank Werner Schindler of BSI for helpful conversations.

References

1. *Digital Signature Standard (DSS)*, FIPS PUB 186-2 (Appendix 6), U.S. National Institute of Standards and Technology, 27 Jan 2000.
2. E. Brier & M. Joye, *Weierstraß Elliptic Curves and Side-Channel Attacks*, Public Key Cryptography (Proc. PKC 2002), D. Naccache & P. Paillier (editors), LNCS **2274**, Springer-Verlag, 2002, pp. 335–345.
3. P. J. Green, R. Noad & N. Smart, *Further Hidden Markov Model Cryptanalysis*, Cryptographic Hardware and Embedded Systems – CHES 2005, LNCS **3659**, Springer-Verlag, 2005, pp. 61–74.

4. 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. K. Koç & C. Paar (editors), LNCS **2523**, Springer-Verlag, 2002, pp. 551–563.
5. C. Karlof & D. Wagner, *Hidden Markov Model Cryptanalysis*, Cryptographic Hardware and Embedded Systems – CHES 2003, C. D. Walter, Ç. K. Koç & C. Paar (editors), LNCS **2779**, Springer-Verlag, 2003, pp. 17–34.
6. P. Kocher, *Timing Attack on Implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology – Crypto '96, N. Koblitz (editor), 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 (editor), 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* Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. K. Koç, D. Naccache & C. Paar (editors), LNCS **2162**, Springer-Verlag, 2001, pp. 391–401.
9. E. Oswald & M. Aigner, *Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks*, Cryptographic Hardware and Embedded Systems – CHES 2001, Ç. K. Koç, D. Naccache & C. Paar (editors), LNCS **2162**, Springer-Verlag, 2001, pp. 39–50.
10. L. R. Rabiner & B. H. Juang, *An Introduction to Hidden Markov Models*, IEEE ASSP Magazine, vol. **3**, no. 1, January 1986, pp. 4–16.
11. A. J. Viterbi, *Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm*, IEEE Trans. Information Theory, vol. **13** no. 2, April 1967, pp. 260–269.
12. C. D. Walter, *Breaking the Liardet-Smart Randomized Exponentiation Algorithm*, Proc. Cardis '02, San José, November 2002, Usenix Association, Berkeley, 2002, pp. 59–68.
13. C. D. Walter, *Issues of Security with the Oswald-Aigner Exponentiation Algorithm*, Topics in Cryptology – CT-RSA 2004, T. Okamoto (editor), LNCS **2964**, Springer-Verlag, 2004, pp. 208–221.
14. C. D. Walter, *Longer Randomly Blinded RSA Keys may be Weaker than Shorter Ones*, Workshop on Information Security Applications – WISA 2007, S. Kim, M. Yung & H.-W. Lee (editors), LNCS **4867**, Springer-Verlag, 2007, pp. 303–316.
15. S.-M. Yen, C.-N. Chen, S. J. Moon & J. C. Ha, *Improvement on Ha-Moon Randomized Exponentiation Algorithm*, Information Security and Cryptology – ICICS 2004, C. Park & S. Chee (editors), LNCS **3506**, Springer-Verlag, 2005, pp. 154–167.

11 Appendix 1: A Markov Model

This section describes the construction of a hidden Markov Model $\mathcal{H}$ for dealing with multiple traces [10]. We start with the case of a single trace.

The recoding automaton is a Markov process with a finite number of states and transitions which depend on the next key digit and bits from a random number generator. On entering a state after traversing the appropriate transition, the recoding algorithm generates a recoded digit which is transformed into a sequence of multiplicative operations. The attacker observes these operations

with restricted clarity. Because the observations do not correspond directly to the states, the sequence of states is not known, giving a *hidden* Markov process.

Without loss of generality, we can assume that the sequence of states determines D uniquely. Then the problem is to determine the most likely sequence of states which generates the given sequence of observations. An algorithm for finding an optimal solution is due to Viterbi [11]. It computes the maximum probability $P_i(s_i)$ of any sequence of states $s_0 s_1 ... s_i$ from the start state s_0 to state s_i at the time of the ith observation, given the sequence of observations up to that point. By keeping track of which state s_{i-1} leads to $P_i(s_i)$, the optimal sequence of states can be reconstructed from the final best state.

The model $\mathcal{H}$ for many traces is constructed as follows. Assume there are $|T|$ copies of the recoding automaton indexed by the elements of the trace set T. These generate the operation sequences that are observed through the traces. Let S be the set of states of the automaton. When i digits of D have been processed by each copy of the automaton, we have an element of $S{\times}\mathbb{N}$ for each $t{\in}T$, which provides the state s_{ti} reached by the automaton of index t, and the number of operations n_{ti} so far in its recoding. This T-tuple of pairs is a state in $\mathcal{H}$. The start state has $(s_{t0}, n_{t0}) = (s_0, 0)$ for each $t{\in}T$. So the state set of $\mathcal{H}$ is the subset of $(S{\times}\mathbb{N})^T$ which is reachable from the start state. This is finite because each n_{ti} is bounded above by the length of its trace t.

Transitions in $\mathcal{H}$ are T-tuples of transitions from the basic recoding automaton, subject to the consistency requirement that they all correspond to the same input bit (or digit). So the inputs which determine a transition in $\mathcal{H}$ are a digit from D and a T-tuple of random numbers. The probability of this transition is the product of the probabilities of the $|T|$ constituent transitions of the original automaton. When the end-of-list symbol $\bullet$ is read from D, each finite automaton enters its final state, and the final state $\bullet$ of $\mathcal{H}$ is reached. The transition to this state generates any necessary final operations in the $|T|$ recoding sequences.

A path $p = s_0 s_1 ... s_i$ in $\mathcal{H}$ represents the first i recoding steps which have been performed for each of the traces. So p determines a path $p_t = s_{0,t} s_{1,t} ... s_{i,t}$ in copy $t \in T$ of the recoding automaton. If r_t is the recoding sequence along that path then it has a metric value $\mu(r_t, t)$ defined in §5. This leads to defining a path metric $\mu(p) = \sum_{t \in T} \mu(r_t, t)$. The "goodness" value of a state is the minimum $\mu(p)$ over all paths p to that state. It is easily computed incrementally by increasing path length. By keeping a pointer back to the previous state on the minimum path, the best path from start to final state can be constructed, and hence the best-fit key obtained.

If the number of operations is completely determined by the key so that all traces have the same length, then there are typically $O(|S|^{|T|})$ states which need processing for each input digit. There are more when trace lengths can vary. So, being exponential in $|T|$, the usual Viterbi algorithm becomes totally impractical when the leakage is so weak that more than a few tens of traces are needed. The algorithm in the main body of the paper is a pruned and re-organised version of this which is linear in $|T|$.

12 Appendix 2: Errors in Attacking Ha-Moon

In the Ha-Moon algorithm [4], the recoding automaton reads one bit at a time and the recoding state is determined by a borrow of 0 or 1. If either the guessed bit or the borrow is 1, but not both, then a given recoding can be extended in two ways: either the next re-coding digit is -1 with borrow 1, or it is $+1$ with borrow 0. Both result in a squaring and a multiplication and hence give rise to the same new metric value and the same new position along the trace, but they differ in the borrow value.

Now select a level l node N in the search tree where this property holds for the best recoding of every trace. These recodings occur in pairs with complementary borrow values. Suppose b_λ is the λ-bit label on the branch from N down to a level $l+\lambda$ node which is labelled with the minimum value $\bar{\mu}_N$ of the metric μ. This minimum arises from taking a good recoding of the prefix D_N for each trace t and appending the best recoding of $b_\lambda+b_t$ where b_t is the existing borrow for the recoding. However, using the complementary λ-bit sequence $2^\lambda-1-b_\lambda$ and complementary borrows $1-b_t$ of the other recodings in each pair, we can obtain the same value of the metric for each trace, and therefore achieve the same minimum $\bar{\mu}_N$ at the complementary level $l+\lambda$ node. This is because, for each trace t, $2^\lambda-b_\lambda-b_t$ has a recoding with the same pattern of squares and multiplications as the chosen recoding for $b_\lambda+b_t$. Specifically, interchanging digits $+1$ and -1 in a recoding of $b_\lambda+b_t$ will give a recoding of $2^\lambda-b_\lambda-b_t$ with exactly the same pattern[5] (and the complementary overflow borrow). Consequently, we obtain best metric values at level $l+\lambda$ from descending the branches corresponding to both b_λ and $2^\lambda-1-b_\lambda$. As one is odd and the other even, we don't know whether the next bit for the best choice should be 0 or 1 – both are equally likely.

If the wrong bit is chosen, the subsequent bits are all wrong until the next point at which the same problem arises. This is because, as in the branch from level l to level $l+\lambda$, the algorithm will continue to generate the best patterns of squares and multiplications, but now by choosing complementary bits, borrows and digits to those which would have been derived had the error not been made.

A consequence of this is that, if no other errors are made, roughly half the bits of the best fit guess at D are incorrect. They occur in sequences of consecutive bits, with changes occurring at predictable points, namely those for which the metric is totally inconclusive about the next bit.

[5] The other digits in the recoding are all 0, and they are the same for both recodings.