Fault Injection and a Timing Channel on an Analysis Technique

John A Clark and Jeremy L Jacob

Dept. of Computer Science, University of York,
York YO10 5DD, England UK
{jac,jeremy}@cs.york.ac.uk

Abstract. Attacks on cryptosystem *implementations* (e.g. security fault injection, timing analysis and differential power analysis) are amongst the most exciting developments in cryptanalysis of the past decade. Altering the internal state of a cryptosystem or profiling the system's computational dynamics can be used to gain a huge amount of information. This paper shows how fault injection and timing analysis can be interpreted for a simulated annealing attack on Pointcheval's Permuted Perceptron Problem (PPP) identification schemes. The work is unusual in that it concerns fault injection and timing analysis on an *analysis technique*. All recommended sizes of the PPP schemes are shown to be unsafe.
Keywords: Heuristic Optimisation, timing channels, identification schemes

1 Introduction: Zero Knowledge and NP-Hard Solution Techniques

Since the introduction of zero-knowledge proofs by Goldwasser et al. in 1985 [5] several schemes have been proposed. Some make use of number theoretic results [3]. Others have sought to make use of the computational intractability of known NP-complete problems. Since Shamir exemplified the concept using the Permuted Kernel Problem (PKP)[12], others have offered systems based on Syndrome Decoding [14], Constrained Linear Equations (CLEs) [15] and the Permuted Perceptron Problem (PPP) [11].

Heuristic optimisation techniques such as genetic algorithms [4] and simulated annealing [7] have shown their worth over a huge number of engineering disciplines. It comes as no surprise that they have been investigated as cryptanalysis tools. Most work has been concerned with elementary ciphers [16, 6, 10, 13] but recent work has included attacks on full-strength zero knowledge schemes. Pointcheval [11] gives results of attacks using simulated annealing on his PPP-based schemes. Knudsen and Meier [8] have recently improved on those results using an unusual and sophisticated attack. Their work (based on properties of sets of solutions obtained from multiple runs) is a direct challenge to the 'standard' way of applying heuristic optimisation techniques and indicates that there is considerable room for more sophistication in their application to cryptanalysis.

This paper describes two further very non-standard approaches based loosely around extant cryptanalysis notions of security fault injection [1] and timing analysis [9]. For comparison, Pointcheval's schemes are the subject of attack. In Section 3 Problem Warping (an interpretation of security fault injection) is used to attack the Perceptron Problem (PP). Cost functions whose minimisation is highly unlikely to lead to an actual solution are used. In a sense, the search is used to find a solution to a warped (different but related) problem. It turns out that the solutions obtained in this way are highly correlated with the solution to the actual problem (much more so than solutions obtained by attempting to solve the original problem directly). Furthermore, combining solutions from differently warped problem searches also provides an efficient way of obtaining secret information. In Section 4 the same technique is used to attack the more difficult Permuted Perceptron Problem (PPP). In addition, a form of timing side-channel with huge power is also demonstrated. The search process is monitored as it moves to its final solution. Some elements of the solution take particular values early in the search and then never change. Observing *when* particular solution elements get 'stuck' in this way can reveal over half the secret vector in a single run. The authors believe that this is the first time *the computational dynamics of an analysis technique* have been used to reveal information. Section 5 draws conclusions and indicates future work. First, the particular schemes analysed are described together with an outline of simulated annealing.

2 Preliminaries

2.1 The Perceptron Problem and Permuted Perceptron Problem

In 1995 Pointcheval [11] suggested what seems a promising scheme based on the *Perceptron Problem* (PP). In fact, he chose a variant of this problem that is much harder to solve known as the *Permuted Perceptron Problem* (PPP). If instances of these problems can be solved the identification schemes are broken. The protocols used to implement the identification schemes are not described here (the reader is referred to [11] for details). This paper concentrates on attacking the underlying NP-complete problems. The notation of [11] and [8] will be used. A column vector whose entries have value +1 or -1 is termed an ϵ -vector. Similarly, a matrix whose entries have value +1 or -1 is termed an ϵ -matrix.

- **Perceptron Problem:** :
  Input: An m by n ϵ-matrix A.
  Problem: Find an ϵ-vector V of size n such that
  $(AV)_i \geq 0$ for all $i = 1, ..., m$.

- **Permuted Perceptron Problem:**
  Input: An m by n ϵ-matrix A and a multiset S of non-negative numbers of size m.
  Problem: Find an ϵ-vector V of size n such that
  $\{\{(AV)_i | i = \{1, ..., m\}\}\} = S$.

In the PP we require that image elements $(AV)_i$ be non-negative, in the PPP we require that these elements have a particular distribution (histogram). If n is odd (even) then the $(AV)_i$ must all take odd (even) numbered values. Pointcheval's PPP schemes used only odd values for n (see below). It is always possible to generate feasible instances of these problems. The matrix A and column vector V are generated randomly. If $(AV)_i < 0$ then the elements a_{ij} of the ith row are negated. This method of generation introduces significant structure into the problem. In particular, the majority vectors of the entries for columns of A are correlated with the corresponding elements of V (as indicated in [11] and [8]). The security of the scheme relies on the computational intractability of exploiting this structure.

Any PPP solution is obviously a solution to the corresponding PP since the PPP simply imposes an extra histogram (multiset) constraint. A solution to the PP is not necessarily a solution to a related PPP. Pointcheval investigated the complexity of generating PPP solutions by the repeated generation of PP solutions. He indicated that matrices of the form $(m, n) = (m, m+16)$ gave best practical security and offered three particular sizes: $(101, 117)$, $(131, 147)$ and $(151, 167)$.

2.2 Simulated Annealing

Simulated annealing is a combinatorial optimisation technique based loosely on the physical annealing process of molten metals. An informal description is given below followed by a detailed one.

States and State Cost. Candidate solutions to the problem at hand form the states over which the search will range. With each state V is associated a cost, $cost(V)$, that gives some measure of how undesirable that state is (in physical annealing a high energy state is undesirable). In attacking the Perceptron and Permuted Perceptron Probems, the current state (solution) will be some ϵ-vector V_{curr} of size n. The choice of cost function is a crucial issue (discussed in Section 3).

The Neighborhood. The search moves from state to state in an attempt to find a state V_{best} with minimum cost over all states. The search may move only to another state that is 'close to' or 'in the neighborhood of' the current one, i.e. it is a *local* search. If V_{curr} is the current state vector, then the local neighborhood $Neighborhood(V_{curr})$ is the set of ϵ-vectors of size n obtained from V_{curr} by negating a single element (i.e. changing a 1 to a -1 or vice versa).

Accepting and Rejecting Moves. Simulated annealing combines hill-climbing with an ability to accept worsening state moves to provide for escape from local optima. From the current state V_{curr} a neighbouring state V_{neigh} is generated randomly. If $cost(V_{neigh}) < cost(V_{curr})$ then V_{neigh} becomes the current state (this is the 'hill-climbing', though perhaps 'valley diving' would be a better term for minimisation problems.) If not, then the state may be accepted probabilistically in a way that depends on the *temperature* T of the search (see below) and the extent to which the target state is worse (in terms of cost). The worse a target state is, the less likely it is a move to that state will be taken.

Cooling It All Down. In analogy with the physical annealing process, simulated annealing has a control parameter T, known as the temperature. Initially the temperature is high and virtually any move is accepted. Gradually the temperature is cooled and it becomes ever harder to accept worsening moves. Eventually the process 'freezes' and only improving moves are accepted at all. If no move has been accepted for some time then the search halts. We now describe the algorithm in detail.

The technique has the following principal parameters:

- the temperature T
- the cooling rate $\alpha \in (0, 1)$
- the number of moves N considered at each temperature cycle
- the number $MaxFailedCycles$ of consecutive failed temperature cycles (where no move is accepted) before the search aborts
- the maximum number IC_{Max} of temperature cycles considered before the search aborts

The initial temperature T_0 is obtained by the technique itself. The other values are typically supplied by the user. In the work described here they remain fixed during a run. More advanced approaches allow these parameters to vary dynamically during the search. The simulated annealing algorithm is as follows:

1. Let T_0 be the start temperature. Increase this temperature until the percentage of moves accepted within an inner loop of N trials exceeds some threshold (e.g. 95%).
2. Set $IC = 0$ (iteration count), $finished = false$ and $ILSinceLastAccept = 0$ (number of inner loops since a move was accepted) and randomly generate an initial current solution V_{curr}.
3. while(not $finished$) do 3a-3d
   (a) **Inner Loop:** repeat N Times
      i. $V_{\text{new}} = \text{generateMoveFrom}(V_{\text{curr}})$
      ii. calculate change in cost
      $\Delta_{\text{cost}} = \text{cost}(V_{\text{new}}) - \text{cost}(V_{\text{curr}})$
      iii. If $\Delta_{\text{cost}} < 0$ then accept the move, i.e. $V_{\text{curr}} = V_{\text{new}}$
      iv. Otherwise generate a value u from a uniform(0,1) random variable. If $\exp^{-\Delta_{\text{cost}}/T} > u$ then accept the move, otherwise reject it.
   (b) if no move has been accepted in most recent inner loop then
   $ILSinceLastAccept = ILSinceLastAccept + 1$
   else $ILSinceLastAccept = 0$
   (c) $T = T * \alpha$, $IC = IC + 1$
   (d) if (ILSinceLastAccept > MaxFailedCycles) or ($IC > IC_{max}$) then
   $finished = true$
4. The state V_{best} giving the lowest cost over the whole search is taken as the final 'solution'.

Note that as T decreases to 0 then $\exp^{-\Delta_{\text{cost}}/T}$ also tends to 0 if $\Delta_{\text{cost}} > 0$ and so the chances of accepting a worsening move become vanishingly small as the temperature is lowered. In all the work reported here, the authors have used a value 0.95 for the geometric cooling parameter α and a value of 400 for N.

In attacking the PPP it will be useful also to record *when* each solution element (i.e. $V_0, \ldots, V_{n-1}$) changed for the last time. For current purposes, the time of last change to an element V_k is deemed to be the index IC of the inner loop in which the last change was made to that element (i.e. when a neighboring solution was obtained by flipping V_k and a move to that solution was taken). These times form the basis of the timing channel indicated earlier.

3 Attacking the Perceptron Problem by Problem Warping

Both Pointcheval [11] and subsequently Knudsen and Meier [8] have attacked the PP using a cost function of the form.

$$Cost(V') = \sum_{i=1}^{m} max\{-(AV')_i, 0\}$$

Pointcheval uses the annealing process directly to obtain solutions to the PP and reports 'We have carried out many tests on square matrices ($m = n$) and on some other sizes, and during a day, we can find a solution to any instance of PP with size less than about 200.' Knudsen and Meier use an iterative procedure, each stage using multiple runs of the annealing algorithm. At each stage commonality between solutions is determined and then fixed for subsequent stages. They report obtaining solutions for various sizes including $(m, n) = (151, 167)$ far quicker (a factor of 180) than those reported earlier.

In both cases the cost function used is very *direct*. It is an obvious characterisation of what the search process is required to achieve. Direct cost functions are, however, not always the most effective. Examination of the way problem instances are generated reveals that small values of $w_i = (AV)_i$ are more likely than larger values. The initial distribution of $(AV)_i$ is (essentially) binomial, with values ranging (potentially) from $-n$ to n. The negation of particular matrix rows simply folds the distribution at 0. This causes difficulties for the search process since attempting to cause negative w_i to become positive by flipping the value of some element V_j is likely to cause various small but positive w_k to become negative. It is just too easy for the search to get stuck in such local optima. One solution is to encourage the w_i to assume values far from 0. This is easily effected: rather than punish when w_i is negative, punish when $w_i < K$ for some positive value K, i.e. use a cost function of the form

$$Cost(V') = g \sum_{i=1}^{m} \left(max\{K - (AV')_i, 0\}\right)^R$$

The cost function is *a means to an end.* A 'good' cost function is one that 'works', i.e. one that guides the search to obtain desired results. Choice of cost

	0	1	2	3		0	1	2	3
Pr 0	0	0	0	3	Pr 5	0	4	6	5
Pr 1	3	6	2	11	Pr 6	3	6	12	5
Pr 2	1	11	6	8	Pr 7	4	7	14	2
Pr 3	8	12	6	3	Pr 8	3	14	2	9
Pr 4	0	4	5	4	Pr 9	1	1	5	4
PP(201,217):30 Runs									

	0	1	2		0	1	2
Pr 0	0	0	1	Pr 5	0	1	0
Pr 1	0	0	2	Pr 6	1	2	6
Pr 2	0	0	1	Pr 7	0	11	6
Pr 3	1	4	14	Pr 8	0	2	9
Pr 4	1	3	6	Pr 9	3	12	11
PP(401,417):40 Runs							

Table 1. Number of successes after simulated annealing plus {0,1,2,3}-bit hill-climbing for (201,217) and (401,417) instances

	0	1	2	3		0	1	2	3
Pr 0	0	0	0	0	Pr 5	0	0	0	2
Pr 1	0	0	1	1	Pr 6	0	0	0	1
Pr 2	0	2	2	4	Pr 7	0	0	0	1
Pr 3	0	1	1	3	Pr 8	0	0	0	0
Pr 4	0	0	0	0	Pr 9	0	1	3	4
PP(501,517):10 Runs									

	0	1	2	3		0	1	2	3
Pr 0	0	0	0	1	Pr 5	0	0	2	2
Pr 1	0	0	0	1	Pr 6	0	2	1	1
Pr 2	0	0	0	0	Pr 7	0	0	0	0
Pr 3	0	0	0	2	Pr 8	0	0	0	0
Pr 4	0	0	0	0	Pr 9	0	0	0	0
PP(601,617):10 Runs									

Table 2. Number of successes after simulated annealing plus {0,1,2,3}-bit hill-climbing for (501,517) and (601,617) instances

function is a subtle matter and experience with many domains suggests that experimentation is essential. This explains the inclusion of R as a parameter. There would seem no a priori reason to restrict R to 1.0 (the value used by previous researchers), and the work reported below shows that higher values give very good results. Here g magnifies the effect of changes when the current solution is changed and is really intended as a weighting factor when the cost function is extended for the PPP problem (see Section 4).

By varying the parameters of this cost function quite radical improvements can be brought in effectiveness. Experiments were carried out for problem instances of sizes $(201, 217)$, $(401, 417)$, $(501, 517)$ and $(601, 617)$. For the $(201, 217)$ problems three values of K were used: $20, 15, 10$. For the $(401, 417)$ problems four values of K were used: $30, 25, 20, 15$. For the rest $K = 25$ was used. In the $(201, 217)$ and $(401, 417)$ cases $R = 2.0$ and in the others $R = 3.0$. A weighting value $g = 20$ was used throughout. For each configuration of parameters ten runs were carried out for each problem instance.

The results are shown in Tables 1 and 2. It was found that there were few direct simulated annealing solutions of the largest PP instances. However, it was often found that flipping a small number of annealing solution bits (e.g. 1, 2 or 3) provided a solution to the PP instance.

All $(201, 217)$ problem instances gave rise to some solution with Problem 0 being the most resilient (only three out of thirty annealing solutions gave rise to a PP solution and then only after three-bit enumerative search). Four of the ten $(401, 417)$ problems produced direct (0-bit search) simulated annealing solutions.

All problems were solved by some run followed by at most an enumerative 2-bit search. For the $(501, 517)$ problems seven produced a solution (with up to 3-bit search used). Half the $(601, 617)$ problems gave rise to a solution. No claim to optimality is made here. For the larger problem sizes only one cost function has been used and then with only ten runs for each problem. The results serve as a simple demonstration of how small changes may matter greatly. The use of cost functions whose minimisation does not lead to the required solution we term *Problem Warping*. The solutions obtained by the warping are highly correlated with the actual defining solution of the problem. For the $(201, 217)$ problems, the best solution over the 30 runs for each problem ranged from 79.2% – 87.1 % correct. For the $(401, 417)$, $(501, 517)$ and $(601, 617)$ problems, the ranges were 83.4 % – 87.5%, 80.6% – 86.4% and 77.5% – 86.1%. This is generally much better than solutions obtained using the standard cost function ($K = 0$ and $R = 1$).

That an enumerative search should be required to obtain PP solutions is not surprising. The cost functions used do not define what it means to be a solution to the PP and the annealing has attempted to solve the problem it was posed. However, the results show that the cost functions used do characterise in some way what it means to be 'close' to a PP solution. The enumerative search can be considered as a second stage optimisation with respect to the traditional cost function (i.e. $K = 0$). The authors would suggest a playful guideline for optimisation researchers in cryptography — if you cannot solve a problem, solve a different one. It might just help!

Application of Problem Warping has allowed instances of the Perceptron Problem with secret vectors **three times longer** than hitherto handled in the literature (an increase in secret state space from around 2^{200} to 2^{617}). This is a huge increase in power and stresses how fragile is current understanding of the power of heuristic optimisation for cryptanalysis (including our own). However, the real power of Problem Warping will be seen in the next section. Its real power lies in its application to the Permuted Perceptron Problem.

4 Attacking the Permuted Perceptron Problem

In 1999 Knudsen and Meier [8] showed that the $(m, n) = (101, 117)$ schemes recommended by Pointcheval were susceptible to a sophisticated attack based on an understanding of patterns in the results obtained during repeated runs of an annealing process. Essentially, their initial simulated annealing process is the standard one (with the number of trials at each temperature cycle equal to n) but with a modified cost function given by

$$Cost(V') = g \sum_{i=1}^{m} \left(max\{K - (AV')_i, 0\}\right)^R + \sum_{i=1}^{n} \left(|H(i) - H'(i)|\right)^R$$

$H(k) = \#\{j : (AV)_j = k\}$, i.e. the number of the $w_i = (AV)_i$ that have value k. H is the reference histogram for the target solution V (i.e. the histogram of the values in AV). Similarly H' is the histogram for the current solution V'.

The histograms apply only to positive $(AV)_i$ elements. In all the experiments reported in [8] $R = 1.0$ and $K = 0$. There, repeated runs are carried out and commonality of the outputs from these runs is noted. Loosely speaking if all runs of the technique agree on certain secret element values there is a good chance that the agreed values are the correct ones. Agreed bits are fixed and the process carried out repeatedly until all bits agree. Unfortunately some (small number of) bits unanimously agreed in this way are actually wrong, and an enumerative search is made for these bits.

4.1 ClearBox Cryptanalysis - Looking Inside the Box

Virtually all applications of optimisation techniques in cryptography view optimisation as a black box technique. A problem is served as input, the optimisation algorithm is applied, and some output is obtained (a candidate secret in the PP and PPP examples). However, in moving from starting solution to eventual solution the heuristic algorithm will have evaluated a cost function at many (possibly hundreds of) thousands of points. Each such evaluation is a source of information for the guidance process. In the black-box approach this information is simply thrown away. For the PPP, the information loss is huge.

As the temperature cools in an application of simulated annealing it becomes more difficult to accept worsening moves. At some stage an element will assume the value of 1 (or -1) and then never change for the rest of the search, i.e. it gets stuck at that value. It is found that some bits have a considerable tendency to get stuck earlier than others when annealing is applied. (Indeed this observation is at the root of Chardaire et al.'s variant of annealing known as thermo-statistical persistency [2].) One could ask 'Why?'. The answer is that the structure of the problem instance defined by the matrix and reference histogram exerts such influence as to cause this. The bits that get stuck early **tend to get stuck at the correct values**. Once a bit has got stuck at the wrong value it is inevitable that other bits will subsequently get stuck at wrong values too. However, it is unclear how many bits will get stuck at the right value before a wrong value is fixed. This has been investigated for various problem sizes and cost functions. Three problem sizes were considered as shown in Table 3. For each problem size a cost function is defined by a value of g, a value of K and a value of R. Thirty problem instances were created for each problem size. For each problem and each cost function ten runs of the annealing process were carried out. The runs were assessed on two criteria: number of bits set correctly in the final solution and number of bits initially stuck correctly before a bit became stuck at an incorrect value.

Thus, for $(101, 117)$ instances there were $3 \times 8 \times 3 = 72$ cost functions and so 720 runs in total for each problem. The results are shown in Table 4. For each problem the maximum number of correctly set bits in a final solution (i.e. the final result of an annealing run) is recorded together with the maximum number bits fixed correctly in a solution before a bit was set incorrectly (usually these will not be simultaneously achieved by one single solution).

(m,n)	Values of g_1	Values of K	Values of R
(101,117)	20,10,5	1,3,5,7,9,11,13,15	2,1.5,1
(131,147)	20,10	7, 10, 13, 16	2,1
(151,167)	20,15,10,5	5, 10, 15, 20	2,1

Table 3. Cost function parameter values. All combinations of g, K and R were used.

Prob	FBC	IBC	Prob	FBC	IBC	Prob	FBC	IBC
Pr 0	102	50	Pr 0	126	42	Pr 0	148	72
Pr 1	100	45	Pr 1	135	68	Pr 1	142	64
Pr 2	103	45	Pr 2	128	64	Pr 2	145	66
Pr 3	99	53	Pr 3	126	672	Pr 3	157	88
Pr 4	101	46	Pr 4	130	39	Pr 4	147	58
Pr 5	108	72	Pr 5	131	70	Pr 5	140	67
Pr 6	99	39	Pr 6	126	47	Pr 6	151	86
Pr 7	101	56	Pr 7	128	56	Pr 7	135	48
Pr 8	104	55	Pr 8	123	52	Pr 8	143	55
Pr 9	106	56	Pr 9	139	75	Pr 9	150	95
Pr 10	102	56	Pr 10	129	51	Pr 10	149	61
Pr 11	107	56	Pr 11	123	48	Pr 11	145	70
Pr 12	101	58	Pr 12	134	57	Pr 12	143	49
Pr 13	104	42	Pr 13	132	62	Pr 13	138	63
Pr 14	102	47	Pr 14	124	37	Pr 14	147	58
Pr 15	102	56	Pr 15	122	59	Pr 15	141	63
Pr 16	101	39	Pr 16	124	41	Pr 16	151	56
Pr 17	103	51	Pr 17	121	42	Pr 17	144	82
Pr 18	103	40	Pr 18	130	62	Pr 18	147	98
Pr 19	103	50	Pr 19	129	53	Pr 19	137	47
Pr 20	105	62	Pr 20	132	67	Pr 20	136	69
Pr 21	107	68	Pr 21	128	59	Pr 21	140	59
Pr 22	106	58	Pr 22	129	97	Pr 22	142	55
Pr 23	103	62	Pr 23	127	61	Pr 23	146	67
Pr 24	103	53	Pr 24	126	43	Pr 24	138	69
Pr 25	100	56	Pr 25	127	72	Pr 25	147	69
Pr 26	104	51	Pr 26	132	44	Pr 26	145	61
Pr 27	98	53	Pr 27	125	68	Pr 27	146	68
Pr 28	105	57	Pr 28	126	38	Pr 28	141	64
Pr 29	103	56	Pr 29	123	50	Pr 29	143	80
Size (101,117) 720 runs			Size (131,147) 160 runs			Size (151,167) 320 runs		

Table 4. Maximum final bits correct (FBC) and maximum initial bits correct (IBC) over all runs. Total number of runs shown for each problem size. Thirty problem instances were attacked for each problem size.

4.2 Making Best Use of Available Information

Consider Ax for any solution vector x. Flipping any single element of x causes the components $(Ax)_i$ to change by ± 2. Similarly, flipping any two bits of x causes the components to change by ± 4 or else stay the same. Flipping three bits causes the components to change by ± 2 or ± 6. Generalising, if x may be transformed into the secret generating solution V by changing an even number of bits, then $(Ax)_i = (AV)_i \pm 4k$ for some integer k. Similarly, if an odd number of bit changes are needed then $(Ax)_i = (AV)_i \pm 4k + 2$. For any x let

$$SUMA(x) = \#\{i : (Ax)_i = 4k + 1, for\ some\ k\}$$

$$SUMB(x) = \#\{i : (Ax)_i = 4k + 3, for\ some\ k\}$$

$SUMA(V) = H(1) + H(5) + \ldots$ and $SUMB(V) = H(3) + H(7) + \ldots$ where H is the publicly available reference histogram. If V is obtained from x by an even number of bit changes, then we have $SUMA(V) = SUMA(x)$ and also $SUMB(V) = SUMB(x)$. If V is obtained from x by an odd number of bit changes, then $SUMA(V) = SUMB(x)$ and $SUMB(V) = SUMA(x)$. Only one of $SUMA(V)$ and $SUMB(V)$ can be odd (since their sum, n, is odd). Thus, for any vector x it is possible to determine whether it differs from V by an even or odd number of bits using the respective values of SUMA(x) and SUMA(V).

Suppose V is the actual secret and x is a solution obtained by annealing. If x is a high performing solution (with few bits wrong) then $(Ax)_i$ will typically be very close to $(AV)_i$. For the (101,117) problem instances, if $(Ax)_i = 1$ then the average actual value of $(AV)_i$ was 6.02. For (131,147) and (151,167) instances the averages were 6.23 and 6.46.

Suppose that $(Ax)_i = 1$ and ten bits are wrong. Typically it will be the case that $(AV)_i \in \{1, 5, 9, 13\}$. This observation has a big impact on enumerative search. For the sake of argument suppose that $(Ax)_i = (AV)_i = 1$. Then flipping the ten wrong bit values to obtain the actual secret must have no effect on the resulting value of $(Ax)_i$. This means that for five wrong bits we must have $a_{ij}x_j = 1$ and for the other five we must have $a_{ij}x_j = -1$. This reduces any enumerative search. For example, searching over 117 bits would usually require C_{10}^{117} (around 4.4×10^{15}) but now requires a search of order around $C_5^{58} \times C_5^{57}$ (around 2.1×10^{13}). This assumes that for solution x $\#\{x_j : a_{ij}x_j = 1\} = 58$ and $\#\{x_j : a_{ij}x_j = -1\} = 57$ (or vice versa). In practice, this may not be the case but any skew actually reduces the complexity of the search. In this respect, it may be computationally advantageous to consider some $(Ax)_i < 0$. For example, if $(Ax)_i = -7$ and there are 10 bits wrong then $(AV)_i$ must be in the range 1..13 with the smaller values much more likely. If $(AV)_i = 1$ then there must be seven wrong bits currently with $a_{ij}x_j = -1$ and three with $a_{ij}x_j = 1$. This is a powerful mechanism that will be used repeatedly.

One has to guess the relationship of $(Ax)_i$ to $(AV)_i$. This will generally add only a factor of about four to the search (and often less). One has also to determine how many bits are actually wrong too. One can start by assuming that the solution vector has the minimum number of bits wrong yet witnessed and engage in enumerative searches. If these fail, simply increment the number of bits assumed incorrect by 2 and repeat the search processes (only even numbers or odd numbers of wrong bits need be considered). The complexity of the search is dominated by the actual number of wrong bits (searches assuming fewer numbers of wrong bits are trivial by comparison). The complexities reported in this paper therefore assume knowledge of the number of wrong bits in the current solution.

4.3 The Direct Attack

It is obvious that 'warping' the cost function produces results that are indeed better than those obtained under the natural cost function. Thus, in the $(101, 117)$ problems three (5, 11 and 22) have given rise to solutions with 10 bits or fewer wrong (from the FBC column of Table 4). Once the highest performing solution has been selected (a factor of 720) an enumerative search of order $C_5^{58} \times C_5^{57}$ (which is less than 2^{45}) will find the solution in these cases. For the $(131, 147)$ and $(151, 167)$ instances extreme results are also occasionally produced. (131,147) Problem 9 gave rise to one solution with only 8 bits wrong. (151,167) Problem 3 similarly gave rise to a solution with only 10 bits wrong. This would require a total search of approximately $320 \times C_5^{84} \times C_5^{83}$ which is less than 2^{60}. Thus, even a fairly brutal search will suffice on occasion, even for the biggest sizes. This is not the most efficient way of solving the problem however.

4.4 Timing Supported Attack

The largest number of initially settled bits can clearly leak a huge amount of information. For (151,167) problems 18, 9 and 3 some solution was obtained whose first 98, 95,88 initially stuck bits were correct. The respective complexities of brute force search over the remaining entries would be of order $2^{69}, 2^{72}, 2^{79}$. Although not within the traditional 2^{64} distance they are sufficiently close to render use of the PPP scheme impossible. For (131,147) there would appear to be an outlier problem 22 with 97 initial bits correct. This leaves a search of order 2^{50}.

Another approach would be to consider in turn all possible pairs of solutions obtained. One pair contains a solution VMAX with the maximum number of bits correct and a solution VINIT with the maximum number of initial bits correct. This pair could form the basis for the subsequent search and we can calculate the computational complexity of finding the exact solution. Obtaining this pair requires a search factor equal to the number of runs squared.

Assume that at least the first I bits initially fixed in VINIT are correct. Change the corresponding bits in VMAX to agree with those in VINIT. These bits are now excluded from the subsequent search — the search will be over the remaining $n - I$ bits of the modified VMAX. For example, suppose in the $(101, 117)$ case that the best initial solution provides us with at least 37 bits (from Table 4 this applies to all 30 problems). This leaves us with 80 bits over which to conduct the remaining search. Suppose ten wrong bits remain. The

66	71	65	69	69	47	75	65	60	56
63	54	64	64	67	63	71	64	67	64
56	51	56	59	63	66	62	70	58	62
85	59	74	76	77	67	84	77	87	47
76	88	64	67	91	85	89	94	71	76
65	76	57	77	85	72	71	77	86	88
80	94	88	56	86	95	70	111	95	68
81	86	97	100	86	96	78	83	72	108
99	99	97	86	97	83	89	85	95	85

Table 5. Search complexities (log 2) of Timing Supported Attacks on (101,117)(upper), (131,147)(middle) and (151,167)(lower) size schemes. Thirty problem instances at each size.

total complexity of the whole search is now approximately

$$720 \times 720 \times C_5^{40} \times C_5^{40} = 2.24 \times 2^{57}$$

Enumerative searches can be performed under optimistic assumptions and these can be progressively relaxed leading to more complex searches. Assuming the number of initially set bits is known, and the number of bits wrong in the best final solution is known, the complexities of the searches are given in Table 5. We have given some very conservative attacks. Some cost functions are clearly more effective than others and it would be possible to restrict attention to a subset of the set consider so far. For the (151,167) problem size Tables 6 and 7 give summary results for each problem instance (over all cost functions) and for each cost function (over all problem instances).

4.5 Other Attacks

Other attacks are possible. For example taking the majority vector over all solution runs (whatever the cost function) can on occasion leak a great deal of information. Commonality of solution elements of repeated runs is at the heart of Knudsen and Meier's technique. This strategy can be adopted here. If runs agree under widespread problem deformation (i.e. using multiple cost functions) then there is often good cause to believe they agree correctly. Rather than insist on absolute agreement, we can rank the secret bits according to the degree of agreement. Frequently the top ranked bits are correct though this method is somewhat erratic. Table 8 shows the number of top ranked bits correct for each (151,167) problem. Thus, for problem 1 the 41 bits that gave rise to most agreement over all runs (the 320 runs indicated in Table 4) were actually correct. We can see that for problems 3, 9 and 18 the most agreed 78, 87 and 88 bits were correct (in the sense that the majority vector is right). This is very significant since around half of all bits are revealed without any kind of enumerative search being deployed.

It is also possible to add up the sticking times of components over all runs. When these are ranked (the highest being the one that took least aggregate time

Problem	Final Bits Correct			Initial Bits Correct		
	Av.Min	Over.Av	Av.Max	Av.Min	Over.Av	Av.Max
Prob 0	130.84	135.82	140.34	6.84	22.06	43.81
Prob 1	125.66	130.25	136.12	7.41	23.68	43.28
Prob 2	129.31	135.24	140.94	8.78	25.78	47.59
Prob 3	133.16	140.88	147.19	23.72	44.34	66.81
Prob 4	127.53	132.96	138.44	9	24.3	41.25
Prob 5	128.22	132	135.72	9.66	27.13	47.22
Prob 6	136.06	140.78	145.22	12.91	31.31	53.69
Prob 7	116.94	123.18	129.22	8.28	21.33	35.56
Prob 8	129.56	133.32	137.22	5.16	16.9	33.09
Prob 9	135.34	139.48	143.84	25.03	49.25	73
Prob 10	127.34	132.09	136.94	3.84	16.82	34.78
Prob 11	127.91	135.78	141.53	23.66	39.82	57.62
Prob 12	124.69	131.02	137.34	6.19	20.69	35.25
Prob 13	122.69	127.63	132.44	10.66	25.44	42.16
Prob 14	127.91	132.17	137.12	9.91	25.47	43.5
Prob 15	125.25	130.67	134.91	5.59	18.6	35.62
Prob 16	132.75	139.54	145.41	3.84	17.9	38.12
Prob 17	125.91	131.04	136.38	24.12	42.57	61.72
Prob 18	133.78	138.31	143.12	30.06	53.65	72.81
Prob 19	122.94	127.63	132.25	1.09	11.93	28.03
Prob 20	122.16	126.53	131.56	10.25	26.03	43.94
Prob 21	126.62	131.68	136.69	8.72	24.37	43.34
Prob 22	123.28	129.99	135.91	4.06	17.48	36.09
Prob 23	127.34	133.53	139.31	7.25	21.68	40.88
Prob 24	120.28	126.09	131.94	16.78	35.12	54.59
Prob 25	130.16	136.05	140.38	8	25.65	46.06
Prob 26	134.09	138.25	141.94	6.84	23.44	45.97
Prob 27	131.22	136.85	142.19	5.97	23.45	46.22
Prob 28	119.03	125.63	133.38	6.19	20.47	37.62
Prob 29	129.12	134.59	139.47	18.25	37.94	59.44

Table 6. Summary results over all cost functions for the (151,167) problem instances. For each problem instance and cost function the minimum final bits correct, the average final bits correct and the maximum final bits correct over the ten annealing runs were calculated. For each problem instance columns 2–4 record the averages of such results over all cost functions. Columns 5–7 record similar information for the initial bits correct.

Parameters	Final Bits Correct			Initial Bits Correct		
(K,g,R)	Av.Min	Over.Av	Av.Max	Av.Min	Over.Av	Av.Max
(20,20,2)	132.47	136.02	139.23	12.9	29.25	49.97
(20,20,1)	129.93	132.71	135.07	8.47	24.13	40.73
(20,15,2)	132.17	135.82	138.9	14.8	30.57	49.27
(20,15,1)	130.03	132.49	135	9.33	24.86	43.27
(20,10,2)	132.6	135.74	138.57	12.77	29.54	49.1
(20,10,1)	129.97	132.37	134.53	10.3	24.3	41.8
(20,5,2)	132.13	135.8	138.9	13.7	30.71	49.33
(20,5,1)	129.73	132.29	135.07	8.47	24.44	41.97
(15,20,2)	129.7	135.13	139.63	12.23	29.67	48.1
(15,20,1)	129.57	133.67	137.6	10.57	26.64	45.63
(15,15,2)	130.87	135.32	140.1	12.37	30.43	50.7
(15,15,1)	130.1	133.76	137.33	12.1	28.09	46.3
(15,10,2)	129.63	135.15	140.1	12.53	30.85	49.83
(15,10,1)	129.23	133.57	137.3	12.83	29.77	47.9
(15,5,2)	130.5	135.49	140.13	14.9	31.16	51.67
(15,5,1)	129.63	133.67	137.3	10.6	27.61	47.07
(10,20,2)	126.67	133.01	140.23	10.53	28.56	48.87
(10,20,1)	128	134.06	139.47	11.27	28.89	50.6
(10,15,2)	126.6	133.39	140.1	11.53	29.25	47.23
(10,15,1)	128.87	134.17	138.9	12.4	29.28	49.47
(10,10,2)	126.93	133.54	139.73	10.67	28.06	49.3
(10,10,1)	128.57	133.91	139.13	14.27	29.95	47.03
(10,5,2)	126.6	133.43	140.53	12.63	28.5	47.73
(10,5,1)	128.8	134.2	139.73	12.77	29.48	49.27
(5,20,2)	120.4	128.16	135.97	7.23	21.24	38.93
(5,20,1)	122	130.1	138.77	8.87	23.74	43.23
(5,15,2)	120.27	129.18	137	7.97	23.81	44
(5,15,1)	122.5	130.04	137.53	7.47	23.73	43.27
(5,10,2)	121.37	129.17	137.03	7.57	22.54	42.1
(5,10,1)	122.8	130.13	137.2	8.9	24.15	44.6
(5,5,2)	121.23	129.19	136.87	8.13	21.95	40.67
(5,5,1)	122.37	130.22	137.77	8.87	23.75	42.77

Table 7. Summary results for each cost function over all (151,167) problem instances. For each problem instance and cost function the minimum final bits correct, the average final bits correct and the maximum final bits correct over the ten annealing runs were calculated. For each cost function columns 2–4 record the averages of such results over all problem instances. Columns 5–7 record similar information for the initial bits correct.

Probs 0–9	0	41	0	78	0	30	0	0	0	87
Probs 10–19	0	41	0	0	0	39	0	62	88	0
Probs 20–29	0	36	0	0	36	32	33	36	28	46

Table 8. Top N agreed correct for problem instances of size (151,167)

to get stuck) the results can also leak information. In some cases only 1 bit of the first ranked 100 for the (151,167) gave rise to a majority vector component that was incorrect.

5 Summary and Conclusions

We have demonstrated that recent fault injection and timing side channel attacks on cryptosystems may be interpreted in the context of optimisation-based search. The attacks on PP and PPP problems have shown the potential power of such interpretations. We make no claims to optimality for our results; extensive experimentation and profiling would allow more effective and efficient sets of cost functions to be determined.

Virtually all work using optimisation techniques attempts to solve the problem at hand in one go. We believe we should stop expecting optimisation to solve problems in this way, and view optimisation as a means of creating data ('failed' solutions) on which to do cryptanalysis! This suggests a new form of cryptanalysis – the profiling and interpretation of local optima obtained by optimisation-based searches. The authors are currently investigating the use of such techniques to block and public key algorithms. We recommend the area to researchers.

6 Acknowledgements

The authors would like to thank Susan Stepney for comments on a previous draft. The authors would also like to thank the anonymous Eurocrypt 2002 referees for their comments.

References

1. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EuroCrypt '97*, pages 37–51, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1233.
2. P Chardaire, J C Lutton, and A Sutter. Thermostatistical persistency: a powerful improving concept for simulated annealing. *European Journal of Operations Research*, 86:565–579, 1995.
3. A. Fiat and A. Shamir. How to Prove Yourself:Practical Solutions of Identification and Signature Problems. In Ed Dawson, Andrew Clark, and Colin Boyd, editors, *Advances in Cryptology — Crypto '86*, pages 186–194. Springer Verlag LNCS 263, july 1987.
4. D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
5. S. Goldwasser, S. Micali, and C. Rackoff. Knowledge Complexity of Identification Proof Schemes. In *17th ACM Symposium on the Theory of Computing STOC*, pages 291–304. SACM, 1985.
6. Giddy J.P. and Safavi-Naini R. Automated Cryptanalysis of Transposition Ciphers. *The Computer Journal*, XVII(4), 1994.

7. S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.
8. Lars R. Knudsen and Willi Meier. Cryptanalysis of an Identification Scheme Based on the Permuted Perceptron Problem. In *Advances in Cryptology Eurocrypt '99*, pages 363–374. Springer Verlag LNCS 1592, 1999.
9. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - Crypto '96*, pages 104–113, Berlin, 1996. Springer-Verlag. Lecture Notes in Computer Science Volume 1109.
10. Robert A J Mathews. The Use of Genetic Algorithms in Cryptanalysis. *Cryptologia*, XVII(2):187–201, April 1993.
11. David Pointcheval. A New Identification Scheme Based on the Perceptron Problems. In *Advances in Cryptology Eurocrypt '95*. Springer Verlag LNCS X, 1995.
12. A. Shamir. An Efficient Scheme Based On Permuted Kernels. In *Advances in Cryptology — Crypto '89*, pages 606–609. Springer Verlag LNCS 435, 1997.
13. Richard Spillman, Mark Janssen, Bob Nelson, and Martin Kepner. Use of A Genetic Algorithm in the Cryptanalysis of Simple Substitution Ciphers. *Cryptologia*, XVII(1):187–201, April 1993.
14. Jaques Stern. A New Identification Scheme Based On Syndrome Decoding. In *Advances in Cryptology —Crypto '93*, pages 13–21. Springer Verlag LNCS 773, 1997.
15. Jaques Stern. Designing Identification Schemes with Keys of Short Size. In *Crypto '93*, pages 164–173. Springer Verlag LNCS 839, 1997.
16. Forsyth W.S. and Safavi-Naini R. Automated Cryptanalysis of Substitution Ciphers. *Cryptologia*, XVII(4):407–418, 1993.