Secret Agents Leave Big Footprints: How to Plant a Cryptographic Trapdoor, and Why You Might Not Get Away with It

John A. Clark, Jeremy L. Jacob, and Susan Stepney

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

Abstract. This paper investigates whether optimisation techniques can be used to evolve artifacts of cryptographic significance which are apparently secure, but which have hidden properties that may facilitate cryptanalysis. We show how this might be done and how such sneaky use of optimisation may be detected.

1   Introduction

The issue of optimisation is fundamental to cryptography. Pseudo-random bit streams should appear as random as possible, cryptanalysts try to extract the maximum amount of useful information from available data and designers wish to make attacks as difficult as possible. It is not surprising that some security researchers have begun to consider the use of heuristic optimisation algorithms. Both simulated annealing and genetic algorithms have been used to decrypt simple substitution and transposition ciphers [6,8,13,14], and there have been some recent and successful attempts to attack modern day crypto protocols based on NP-hard problems [3]. Recent research has used optimisation for design synthesis, evolving Boolean functions for cryptographic products, most notably using hill climbing [10] and genetic algorithms [9]. Other recent work has applied search to the synthesis of secure protocols [1].

Some cryptographic algorithms have been shown to be insecure. Others are thought to be secure, but subsequent discoveries may prove otherwise. Cryptanalysts live in hope that the ciphers they investigate have as yet undiscovered features that will allow them to be broken, 'trapdoors' if you like, hidden ways of opening up the algorithms to let the secrets of those using them fall out! Discovering such trapdoors is very hard work. There is, however, a somewhat sneakier approach — deliberately create an algorithm with a subtle trapdoor in it but which looks secure according to currently accepted criteria. Users of the algorithm may communicate secrets, but the analyst who knows the trapdoor will be able to access them. If users subsequently find out that a trapdoor had been deliberately engineered into the algorithm, they may not be very happy. The trapdoor should be so subtle that they have little chance of discovering it.

This paper investigates whether optimisation techniques can be used to surreptitiously plant such trapdoors.

This issue goes to the heart of cryptosystem design in the past. The debate about whether the Data Encryption Standard [11] has a secret trapdoor in its substitution boxes [1] has raged since its publication [12]. A major factor in this debate is that initially the design criteria were not made public (and the full criteria are still unknown). There was a distinct suspicion that security authorities had designed in a trapdoor or otherwise knew of one. [2] We contend that combinatorial and other numerical optimisation techniques can be *demonstrably open* or *honest*. Goals are overt (that is, maximising or minimising a specified design function). The means of achieving those ends is entirely clear — the algorithm for design is the search technique and its initialisation data coupled with the specified design function to be minimised. An analyst who doubts the integrity of the design process can simply replay the search to obtain the same result. Furthermore, the approach permits some surprising analyses.

2   Public and Hidden Design Criteria

The set of all possible designs forms a design space D. From within the design space we generally seek one instantiation whose exhibited properties are "good" in some sense. (In the case of DES, one design space may be thought of as the set of legitimate S-box configurations. The standard uses a particular configuration but many others are possible.) When designing a cryptosystem, we may require that certain stringent public properties P be met. Generally only small fraction of possible designs satisfy these properties. Consider now a trapdoor property T. If a design has property T then cryptanalysis may be greatly facilitated. Some designs may simultaneously satisfy P and T. There two possibilities here:

- All designs (or a good number) satisfying P also satisfy T. This suggests that the basic design family is flawed (though the flaw may not be publicly known).
- Only a small fraction of designs satisfying P also satisfy T. Here the trapdoor planter must drive the design in such a way that T is achieved simultaneously with P.

---

[1] DES is a 16-round cipher. Each round is a mini-encryption algorithm with the outputs of one round forming the inputs of the next. Each round application further obscures the relationship to the initial inputs. At the end of 16 rounds, there should be a *seemingly* random relationship between the initial inputs and the final outputs. (Of course, the relationship is not random, since if you have the secret key you can recover the plaintext inputs) Within each round there are several substitution boxes (S-boxes). These take 6 input bits and produce 4-bit outputs. The details need not concern us here. Suffice it to say that S-boxes are designed to give a complex input-output mapping to prevent particular types of attacks.

[2] As it happens, it seems they intervened to make the algorithm secure against differential cryptanalysis, a form of attack that would be discovered by academics only in the late 1980s.
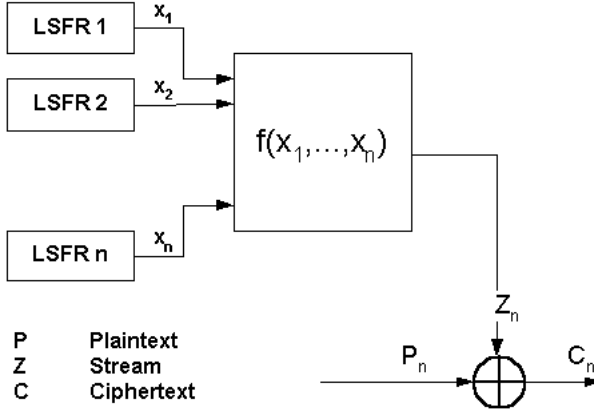
Fig. 1. Combining bit streams

It may be computationally very hard to locate good designs. Heuristic optimisation offers a potential solution. Suppose a cost function motivated solely by the goal of evolving a design (from design space D) to satisfy P is given by $h : D \rightarrow \Re$. (We denote this cost function by h since its choice is *honest* — free from hidden motives.) Suppose also that a cost function aimed at evolving a design satisfying the trapdoor property T is given by $t : D \rightarrow \Re$ (t for trapdoor). If cryptographic functionality is developed using only trapdoor criteria T it is highly likely that someone will notice! Indeed, reasonable performance of the resulting artifacts when judged against public properties P would be accidental. Similarly, an artifact developed using only the honest (public) criteria P will generally have poor trapdoor performance. There is a tradeoff between malicious effectiveness (i.e. attaining good trapdoor properties) and being caught out. The more blunt the malice, the more likely the detection. We capture the degree of bluntness via a parameter λ, which we term the *malice factor*. The design problem for malice factor $\lambda \in [0.0, 1.0]$ is to minimise a cost function of the form

$$cost(d) = (1 - \lambda)h(d) + \lambda t(d) \tag{1}$$

At the extremes we have the honest design problem, $\lambda = 0$, and unrestrained malice, $\lambda = 1$. In between we have a range of tradeoff problems. We now illustrate how optimisation can be used and abused in the development of perhaps the simplest cryptographically relevant element, namely a Boolean function.

3   Cryptographic Design with Optimisation

3.1   Boolean Functions

Boolean functions play a critical role in cryptography. Each output from an S-box can be regarded as a Boolean function in its own right. Furthermore, in

stream ciphers Boolean functions may be used to combine the outputs of several bit streams. Figure 1 illustrates the classic stream cipher model. The plaintext stream $\{P_i\}$ of bits is XOR-ed with a pseudo-random bit stream $\{Z_i\}$ to give a cipher text stream $\{C_i\}$. The plaintext is recovered by the receiver by XOR-ing the cipherstream with the same pseudo-random stream. The pseudo-random stream is formed from several bit streams generated by Linear Feedback Shift Registers (LFSRs) using a suitable combining function. The initial state of the registers forms the secret key. The function f must be sufficiently complex that cryptanalysts will not be able to determine the initial state of the registers, even when they know what the plaintext is (and so can recover the pseudo-random stream $\{Z_i\}$). Two desirable features are that the function should be highly non-linear and posses low auto-correlation. These are explained below, together with some further preliminaries.

We denote the *binary truth table* of a Boolean function by $f : \{0,1\}^n \to \{0,1\}$ mapping each combination of n binary variables to some binary value. If the number of combinations mapping to 0 is the same as the number mapping to 1 then the function is said to be *balanced*. A cryptographic designer may require a Boolean function to have this property. The *polarity truth table* is a particularly useful representation for our purposes. It is defined by $\hat{f}(x) = (-1)^{f(x)}$. Two functions f, g are said to be *uncorrelated* when $\sum_x \hat{f}(x)\hat{g}(x) = 0$. If so, if you approximate f by using g you will be right half the time and wrong half the time.

An area of particular importance for cryptanalysts is the ability to approximate a function f by a simple linear function, that is, one of the form $L_\omega(x) = \omega_1 x_1 \oplus \omega_2 x_2 \ldots \oplus \omega_n x_n$ (where $\oplus$ is exclusive or). For example, $x_1 \oplus x_3 \oplus x_4$ is a linear function (with $\omega = 1011$), whereas $x_1 x_2 \oplus x_3 \oplus x_4$ is not: it has a quadratic term. One of the cryptosystem designer's tasks is to make such approximation as difficult as possible; the function f should exhibit high *non-linearity* [3]. In terms of the nomenclature given above, the formal definition of non-linearity is given by

$$NL(f) = \frac{1}{2}\left(2^n - \max_\omega \left|\sum_x \hat{f}(x)\hat{L}_\omega(x)\right|\right) \tag{2}$$

The 2^n functions $\{\hat{L}_\omega(x), \omega : 0..(2^{n-1})\}$ span the space of Boolean functions on n variables. The term $\sum_x \hat{f}(x)\hat{L}_\omega(x)$ (usually referred to as the Walsh value at ω) can be thought of as the dot product of $\hat{f}$ with $\hat{L}_\omega$. Thus, we are trying to minimise the maximum absolute value of a projection onto a linear function.

---

[3] If it doesn't then various forms of attack become possible, e.g. the Best Affine Attack. Loosely, linear functions are not very complex input-output mappings. If the function $\hat{f}$ can be approximated by a linear function, it is just too predictable to resist attack. See [5] for details.

Low *auto-correlation* is another important property that cryptographically strong Boolean functions should possess. Auto-correlation is defined by

$$AC_{max} = \max_{s \neq 0} \left| \sum_x \hat{f}(x)\hat{f}(x \oplus s) \right| \tag{3}$$

It is essentially a property about dependencies between periodically spaced elements. The details need not concern us here. What is important is that non-linearity and autocorrelation of a given Boolean function f can be measured. For the purpose of this paper we simply identify balancedness, high non-linearity and low autocorrelation to serve as our desirable publicly stated properties P.

3.2   An Honest Cost Function

We aim to provide balanced Boolean functions of 8 variables with high non-linearity and low autocorrelation. We adopt a search strategy that starts with a balanced (but otherwise random) function, and explores the design space by moving between neighbouring functions until an appropriate solution is found. A function $\hat{f}$ can be represented by a vector of length 256 with elements equal to 1 or -1. The first element is $\hat{f}(00000000)$ and the last $\hat{f}(11111111)$ etc. We define the neighbourhood of a balanced function $\hat{f}$ to be all functions $\hat{g}$ obtained from $\hat{f}$ by negating any two dissimilar elements in the vector (that is, changing a 1 to a -1 and -1 to a 1). Provided the initial function was balanced, the search maintains that balance.

To use heuristic optimisation techniques we need a cost function whose minimisation gives good values of non-linearity and low values for autocorrelation. We use the (honest) cost function

$$h(\hat{f}) = \sum_\omega \left| \left| \sum_x \hat{f}(x)\hat{L}_\omega(x) \right| - 12 \right|^3 \tag{4}$$

where ω, x range over $\{0, 1\}^8$. The details of this cost function and the rationale for it can be found in [2]. It gives very good results for Boolean functions of 8 variables; but we admit it is not intuitively clear. There is a tradeoff between a "clearly" honest cost function, and one with good performance.

The primary search technique used was the simulated annealing algorithm [7] with the honest cost function of Equation 4. This was followed by hill-climbing (attempting to maximise directly the non-linearity as defined in Equation 2) from the resulting function $\hat{f}$. Figure 2 shows the results from 400 runs of the technique using $\lambda = 0$, that is, total honesty. 115 of the 400 runs gave rise to functions with non-linearity of 116 and autocorrelation of 32. The highest non-linearity ever attained by *any* technique (ignoring auto-correlation) is 116. Until recently the lowest autocorrelation possible was thought to be 24. Recent optimisation based work [4] generated functions with autocorrelation of 16 (but with nonlinearity value of 112). No function has ever been demonstrated with nonlinearity 116 and autocorrelation 16.

AC_{max}	Non-linearity			
	110	112	114	116
56	0	0	0	0
48	0	2	5	13
40	0	9	68	80
32	0	29	74	115
24	0	1	1	3

Fig. 2. Honest Evolution, sample size = 400. Simultaneous high non-linearity and low autocorrelation (bottom right corner) are desirable.

Our approach appears to give excellent results for both criteria simulataneously. Thus the technique can be used effectively by honest researchers to achieve honest goals. The search proceeded from a random starting function to its final goals guided by the honest cost function at each stage. Now we choose to guide the search in a malicious way.

3.3   Evolving a Trapdoor

We now seek designs that perform well when judged against the public criteria P but also possess some secret trapdoor property T. For illustrative purposes only, we choose to define a trapdoor property T as sufficient closeness to some arbitrary but fixed Boolean function represented by g. We take $\left|\sum_x \hat{f}(x)\hat{g}(x)\right|$ as a measure of closeness. The maximum value 256 occurs when f and g are the same function (or when one is the logical negation of the other). Since our design framework is based around minimising cost, the closer the value to 256 the smaller should be the cost. This gives rise to a trapdoor cost function

$$t(\hat{f}) = \left|256 - \left|\sum_x \hat{f}(x)\hat{g}(x)\right|\right|^3 \tag{5}$$

The exponent 3 is adopted for comparability with the honest cost function. It serves to punish poorly performing functions very highly and affects the rate at which the search converges. We now combine the honest and dishonest functions in a single cost function *cost* as given in Equation 1 (with the function $\hat{f}$ playing the role of design d).

We applied the technique for various values of λ in the range $[0, 1]$. For a randomly generated trapdoor g, 30 runs were carried out for each of $\lambda \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. The same trapdoor was used in all runs. Figure 3 records the results. Thus, for $\lambda = 0.0$, 12 of the 30 runs give rise to functions with nonlinearity of 116 and autocorrelation of 32. The average closeness to the trapdoor g (given by $\left|\sum_x \hat{f}(x)\hat{g}(x)\right|$) for the 30 runs in this case is 12.8.

The results show that optimisation techniques can be used effectively to obtain artifacts that provide good public performance measures and good trapdoor functionality. The figures show that as λ increases the performance of the evolved

AC_{max}	$\lambda = 0.0$ Non-linearity				$\lambda = 0.2$ Non-linearity				$\lambda = 0.4$ Non-linearity			
	110	112	114	116	110	112	114	116	110	112	114	116
64	0	0	0	0	0	0	0	0	0	0	1	0
56	0	0	0	0	0	0	1	0	0	0	2	0
48	0	0	0	1	0	0	7	0	0	1	6	0
40	0	0	3	4	0	0	16	0	0	2	17	0
32	0	2	7	12	0	0	6	0	0	0	1	0
24	0	0	0	1	0	0	0	0	0	0	0	0
	Mean Trap = 12.8				Mean Trap = 198.9				Mean Trap = 213.1			

AC_{max}	$\lambda = 0.6$ Non-linearity				$\lambda = 0.8$ Non-linearity				$\lambda = 1.0$ Non-linearity			
	110	112	114	116	110	112	114	116	110	112	114	116
80	0	0	0	0	0	0	0	0	2	0	0	0
72	0	0	0	0	0	0	0	0	4	1	0	0
64	0	0	1	0	0	1	0	0	10	6	0	0
56	0	1	2	0	0	4	1	0	2	5	0	0
48	0	5	7	0	0	19	1	0	0	0	0	0
40	0	2	12	0	0	3	1	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0
	Mean Trap = 222.1				Mean Trap = 232.3				Mean Trap = 242.7			

Fig. 3. Performance for various malice factors λ. Sample size = 30 for all experiments. For the $\lambda = 1$ result, any remaining non-linearity and auto-correlation arises from the dishonest g and the final hill-climb on non-linearity.

designs with respect to the public criteria worsens (i.e their non-linearity gets smaller and autocorrelation gets bigger). Of course, the closeness of the evolved designs to the trapdoor property gets radically better. An honest sample (that is, $\lambda = 0$) has an average closeness of 12.8. With $\lambda = 0.2$ the average solution obtained has a closeness value of 198.9. With $\lambda = 1.0$ the average is 242.7[4].

4   Detection

If you know what the trapdoor is then you can of course verify that the artifact has it. But what if you don't know what the trapdoor is?

4.1   Telling Truth from Fiction

We wish to distinguish honestly derived functions from maliciously derived ones. We would like to develop discriminating techniques with wide ranging applicability. We observe that functions (and indeed designs more generally) have some

---

[4] Recall that after optimising with respect to the indicated cost function, we then hill-climb (maximise) with respect to non-linearity alone. This typically serves to reduce the trapdoor value attained - it is simply measured at the end of the hill-climb. It plays no part during the hill-climb.

concrete binary representation. Our current function representation is enumerative, but *programs* that implement particular functions also have binary representations (the vector of object code bits being a particular one). A function within a particular family may be defined by some set of configuration data (e.g. S-Boxes). It is the properties of such vector representations of designs that we propose to use as the basis of detection. This has the benefit of being more problem independent.

We illustrate one such way in which families of designs can be discriminated. Suppose we have two groups of designs G_1, G_2. Each group comprises a number of designs represented by vectors. For each group we calculate the mean vector for the vectors in that group. Denote these mean vectors by m_1, m_2. Re-express $m_1 = \mu m_2 + z$, where z is orthogonal to m_2. The first component μm_2 is the projection of m_1 onto m_2. We now project each design onto this residual z and measure the length of such projections. In general if the groups G_1 and G_2 are different we would expect those in G_1 to give rise to projections of large magnitude and those in G_2 to give rise to vectors of small magnitude. More formally

- Let d_{1j} and d_{2j} be the jth designs of G_1 and G_2 respectively. Let d_{1j} be represented by a vector of N bit values $(d_{1jk}), k = 1..N$. Similarly for d_{2j}.
- For each group form the mean vector $m_i = (m_{ik})$, where $m_{ik} = \frac{1}{S}\sum_{j=1}^{D} d_{ijk}$
- Calculate the residual vector when m_1 is projected on m_2. For vectors x, y the residual of x after projection on y is given by $resid(x, y) = x - \frac{(x,y)}{(y,y)}y$, where (x, y) is the 'dot product' for vectors x and y.
- for G_1, G_2 find the dot product of each of its designs onto this residual vector and find its absolute value.

4.2 Results

Figure 4 shows the the summary statistics (with the honest group as the reference sample) for each of the λ values, for 30 runs. The aim is to show the degree to which the summary statistics for each malicious sample overlaps with the set of values from the honest one. For $\lambda = 0$, every value is ≤ 0, by construction. By contrast, for other values of λ, very few results are ≤ 0. For $\lambda = 0.6$ only one design has a result in the range of honest results; for other values of λ the overlap is greater.

We see that the summary statistic provides a reasonable, and in some cases good, way of discriminating between designs developed using different cost functions. In some cases the discrimination capability is not as good as we should like, because the calculated statistic loses a great deal of information. However, even with this gross loss of information we can still detect a discernible difference in malicious and honest designs. Clearly more sophisticated tests are needed.

The above is intended purely as an illustration that families of designs give rise to their own distributions of summary statistics. It should be noted that the designs within a family are not necessarily close to each other. Indeed, they may tend to congregate in different areas (corresponding to different local optima of
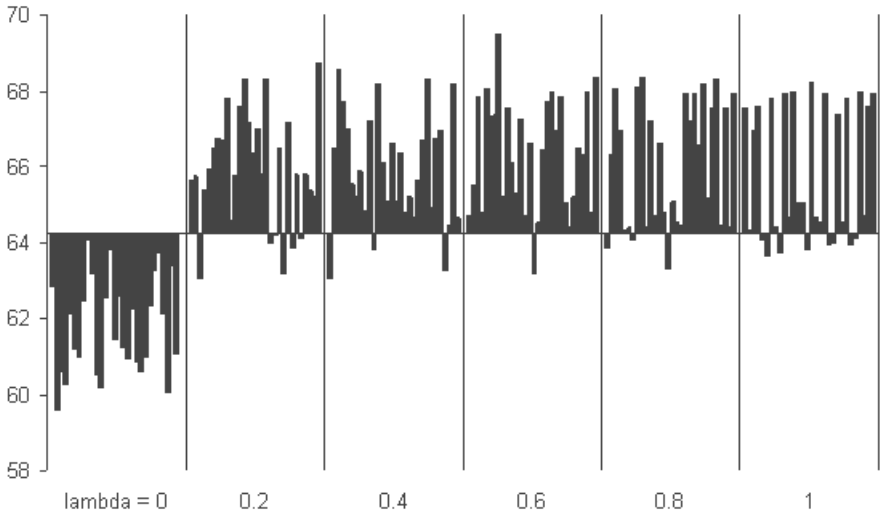
Fig. 4. Squares of projections, for different values of λ. The axis is drawn at the maximum value of the $\lambda = 0$ statistic, to highlight the contrasting behaviour for other values of λ.

cost function used). A malicious designer might take advantage of this weakness or similar weaknesses of other discrimination techniques used. This we now show.

4.3 Games and Meta-games

If the discrimination technique is known then it is possible to recast the malicious design problem with a cost function

$$cost(d) = (1 - \lambda)h(d) + \lambda t(d) + easeDetect(d) \qquad (6)$$

Here *easeDetect* represents how easily the solution may be identified as a maliciously derived one. Given the scheme above a suitable candidate would be to punish solutions whose summary statistic was too large. If a variety of known discriminants are used each of these can similarly be taken into account. Of course, it becomes more difficult to find a good solution, but the malicious designer is really just looking to gain an edge.

Malicious designers can attempt to evade detection in this way only if they know the discrimination tests to be applied. Analysts are free to choose fairly arbitrarily which tests to apply. This makes the task of malicious designers more difficult. They can appeal to their intuition as to what sorts of tests might be sensible (as indeed must the analyst), but they will be second guessing the analyst at large.

4.4   Meta-meta Games

The work described has concentrated on how trapdoors can be inserted and about how claims to have used particular cost functions can be checked. We stated in the introduction that a designer could simply publish the cost function, the search algorithm, and its initialisation data; the doubting analyst could simply replay the search to obtain the same result. This is undoubtedly so, but the implication that this gives us *honest design* must be questioned. We must question in fact what it means to be 'honest'.

Our "honest" cost function of Equation 4 uses an exponent of 3. Why 3? Would 2.9 be dishonest? It really depends on what our *intention* is for choosing a particular value. As it happens we have found that 3.0 gives good results for finding the desired properties, but other values may perform similarly well. Our honest cost function is indeed significantly different from those used by other researchers, whose cost functions are actually more intuitively clear, but seem to perform less well.

It would not be surprising if functions derived using different honest cost functions gave rise to results which have different properties as far as illicit trapdoor functionality is concerned. Large scale experimentation could lead to the discovery of cost functions that give results with excellent public properties but which *happen* to have a good element of specific trapdoor functionality. Such higher level games could be played most easily by those with massive computational capability. Perhaps this could be countered by cost function cryptanalysis!

As a final game, consider the following. Suppose you discover a search technique that works better than existing ones. Suppose, for example, it could find with ease a balanced Boolean function of 8 variables with non-linearity of 118. (This would answer an open research question, since the best least upper bound is 118, but the best actual non-linearity achieved is 116.) If you so desired you could use you enhanced search capability to find functions that had, say, non-linearity 116 but good trapdoor properties too. *You* know that you have sacrificed a little non-linearity to achieve your malicious goal, but the public analyst will compare your results against the best *published* ones. Enhanced search can also be provided by increased computational power. Of course, the malicious designer will have to do without the kudos that would be attached to publishing results to open problems (an academic would of course never do this!).

5   Conclusions

This paper builds on extant optimisation-based cryptographic design synthesis work and extends it to show the optimisation techniques offer the potential for planting trapdoors. We have explored some of the consequences. We know of no comparable work.

Different cost functions solve different problems. Someone might be solving a different problem from the one you assumed; someone may have added a trapdoor to their alleged cost function. However, you can use optimisation to evolve a design against a criterion many times. The resulting *sample* of solutions may

provide a yardstick against which to judge proffered designs. It provides therefore a potential means of resolving whether a trapdoor was present in the fitness function actually used. Currently, it is hard to generate a large sample of good designs by other techniques (without the benefit of automatic search it may be difficult to demonstrate a single good design).

That designs evolved against different criteria should exhibit different functionality is clearly no surprise — it means that optimisation based approach works! This paper has observed that different designs have different bit level representations and that different groups of designs can be detected by using statistical properties of the representations.

The following consequences have emerged from the work:

An optimisation-based design process may be open and reproducible. A designer can publish the criteria (that is, the cost function) and the optimisation search algorithm (together with any initialistion data such as random number seed).The search can be repeated by interested parties.

Optimisation can be abused. If optimisation 'works' then 'use' means one approves of the cost criteria and 'abuse' means one doesn't!

Optimisation allows a family of representative designs to be obtained. The search process may be repeated as often as desired to give a set of designs from which statistical distributions of designs evolved to possess particular properties may be extracted.

Designs developed against different criteria just look different! If designs have different functionality then they are different in some discernible way. In certain cases this difference may be detected by *examination of the design representations alone.*

The games just do not stop. Cryptographic design and analysis go hand in hand and evolve together. When the smart cryptanalyst makes an important discovery, the designers must counter.

These ideas are at an early stage at the moment, but there would appear to be several natural extensions. The simple projection method used for illustration in this paper loses a very large amount of information. A study of available discrimination techniques should bring great benefit. We aim to evolve larger artifacts with trapdoor functionality, and are currently attempting to evolve DES-family cryptosystems with built-in trapdoors (by allowing a search to range over S-Box configurations). We acknowledge that we may have been beaten to it! How could we tell?

6   Coda: The Authors Are $\lambda = 0$ People

Our aim in this paper has been to point out what is *possible* or indeed what might already have been attempted (secretly). We do not wish to suggest that the techniques *should* be used to plant trapdoors. Indeed, this is the reason why we have begun to investigate the abuse of optimisation via discriminatory tests. The paper is, we hope, an interesting, original and rather playful exploration of an intriguing issue.

References

1. John A Clark and Jeremy L Jacob. Searching for a Solution: Engineering Trade-offs and the Evolution of Provably Secure Protocols. In *Proceedings 2000 IEEE Symposium on Research in Security and Privacy*, pages 82–95. IEEE Computer Society, May 2000.
2. John A Clark and Jeremy L Jacob. Two Stage Optimisation in the Design of Boolean Functions. In Ed Dawson, Andrew Clark, and Colin Boyd, editors, *5th Australasian Conference on Information Security and Privacy, ACISP 2000*, pages 242–254. Springer Verlag LNCS 1841, july 2000.
3. John A Clark and Jeremy L Jacob. Fault Injection and a Timing Channel on an Analysis Technique. In *Advances in Cryptology Eurocrypt 2002*. Springer Verlag LNCS 1592, 2002.
4. John A Clark, Jeremy L Jacob, Susan Stepney, Subhamoy Maitra, and William Millan. Evolving Boolean Functions Satisfying Multiple Criteria. In Alfred Menezes and Palash Sarkar, editors, *INDOCRYPT*, volume 2551 of *Lecture Notes in Computer Science*. Springer, 2002.
5. C. Ding, G. Xiao, and W. Shan. *The Stability of Stream Ciphers*, volume 561 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.
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. Robert A J Mathews. The Use of Genetic Algorithms in Cryptanalysis. *Cryptologia*, XVII(2):187–201, April 1993.
9. W. Millan, A. Clark, and E. Dawson. An Effective Genetic Algorithm for Finding Highly Non-linear Boolean Functions. pages 149–158, 1997. Lecture Notes in Computer Science Volume 1334.
10. W. Millan, A. Clark, and E. Dawson. Boolean Function Design Using Hill Climbing Methods. In Bruce Schneier, editor, *4th Australian Conference on Information Security and Privacy*. Springer-Verlag, april 1999. Lecture Notes in Computer Science Volume 1978.
11. National Bureau of Standards. Data Encryption Standard. *NBS FIPS PUB 46*, 1976.
12. Bruce Schneier. *Applied Cryptography*. Wiley, 1996.
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. Forsyth W.S. and Safavi-Naini R. Automated Cryptanalysis of Substitution Ciphers. *Cryptologia*, XVII(4):407–418, 1993.