Applications of search techniques to cryptanalysis and the construction of cipher components.

James David McLaughlin

Submitted for the degree of Doctor of Philosophy (PhD)

University of York
Department of Computer Science
September 2012

Abstract

In this dissertation, we investigate the ways in which search techniques, and in particular metaheuristic search techniques, can be used in cryptology. We address the design of simple cryptographic components (Boolean functions), before moving on to more complex entities (S-boxes). The emphasis then shifts from the construction of cryptographic artefacts to the related area of cryptanalysis, in which we first derive non-linear approximations to S-boxes more powerful than the existing linear approximations, and then exploit these in cryptanalytic attacks against the ciphers DES and Serpent.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Professor John Clark for his advice and supervision during the past four years, and for laying the foundations for the use of metaheuristic search in cryptology. I would also like to thank Miia Hermelin and Kaisa Nyberg for their advice when working on the statistical underpinnings for nonlinear cryptanalysis, Claude Carlet for answering my questions regarding filter functions in stream ciphers, Orr Dunkelman for providing a corrected version of his linear approximation and advising me on complexity issues relating to memory accesses, Howard Heys for providing me with test vectors when I was implementing his toy cipher for use in experiments, Lars Knudsen for answering my questions regarding nonlinear approximations, Qichun Wang for supplying me with the truth table for the nonlinearity 32556 Boolean function, Gregory Bard for supplying me with an advance copy of his book on algebraic cryptanalysis, Enes Pasalic for informing me of the discovery of a 6×6 bijective APN S-box by J.F. Dillon, and Dillon himself for sending me the truth table for the S-box. Without EPSRC's sponsorship none of this research would have been possible, and I would like to thank them for the financial support they provided me during the PhD. Finally, I would like to thank the Computer Science Department's former Research Studies Administrator, Filomena Ottaway, for the invaluable support and assistance she provided to myself, and indeed to all of the other PhD students in the department during her time there.

Author's declaration

This thesis is entirely the work of James David McLaughlin, and was carried out at the University of York between October 2008 and September 2012. No portion of this thesis has previously been formally published elsewhere.

Chapter 1

Introduction.

This thesis involves two separate, and highly distinctive, areas of computer science: cryptology and metaheuristic search.

Metaheuristic algorithms are general-purpose search algorithms that enable complex search spaces to be traversed in search of optimal or high-quality solutions to a given problem. They do not guarantee to find an optimal solution, but provide an efficient means of obtaining very high-performing solutions.

Most metaheuristic search algorithms are based on analogies with natural processes. For example, simulated annealing is based on an analogy with the physical processes of cooling metals, genetic algorithms are based on the processes of natural evolution, memetic algorithms build on these using an analogy to the concept of *memes* as defined by Richard Dawkins [116], and ant algorithms take their inspiration from the foraging techniques used by real ants.

Cryptology, a discipline predating the invention of computers, started out as a means of preventing access to data by those not authorised to view it, as well as the study of means to subvert this protection. In the Computer Age, cryptology has also found several other applications in the field of computer security, such as tamper-proofing important documents and verifying their authorship.

Metaheuristics, while used recreationally to break pre-computer age ciphers, were not applied to serious cryptology until the start of the 21st century, when various papers by (and coauthored by) John Clark [74, 82, 81, 83, 79, 64, 84, 86, 87, 80] demonstrated the use of metaheuristics in the design of communication protocols and Boolean functions for use in cryptology, as well as in the related field of quantum computation.

However, many unsolved problems still remained in all these fields. Furthermore, while this research had focused on *cryptography* - the construction of secure cryptologic algorithms - there had been little work on *cryptanalysis* - the search for weaknesses in these constructions. It may well have seemed counterintuitive that metaheuristics could contribute to this field - the application of cryptography to data was intended to completely obliterate any patterns in the data, leaving no "order amongst the chaos" for the search algorithms to take advantage of. Such a view, however, ignored the possibility that instead of being used to try to reverse cryptographic transforms directly, or to recover protected data, metaheuristics could well be used to analyse the constructions of ciphers, hash functions and other cryptographic algorithms, and identify weaknesses that a cryptanalyst could exploit more conventionally.

This, then is our hypothesis: **That metaheuristic search techniques have a great deal of potential within cryptology; both to improve upon the design of modern-day ciphers and their components, and to contribute to their cryptanalysis.**

1.1 The Structure of this Thesis

This thesis is structured as follows:

Chapter 2 presents a brief overview of the history of cryptology, in particular the astonishing advances made in the second half of the 20th century.

Chapter 3 is the literature review chapter, describing various block cipher constructions and cryptanalysis methods, as well as stream ciphers based on the LFSR constructions. It then introduces the reader to metaheuristics, describing various metaheuristic search algorithms and the application of metaheuristics to both cryptography and cryptanalysis.

Chapter 4 describes the application of metaheuristic search to the design of so-called "filter functions" for use in stream ciphers. It is shown that the important properties of *algebraic immunity* and *fast algebraic resistance*, to which metaheuristics have not so far been applied, can in fact be optimised using "local search"-based metaheuristic algorithms such as simulated annealing, and we seek to simultaneously optimise these and other properties of the filter functions.

Chapter 5 applies metaheuristic search to the design of "S-boxes", components used to ensure nonlinearity in block cipher constructions. Existing research by Clark et al. [86] is built on firstly by focusing on the criterion of *differential uniformity* that was not addressed in earlier work, and secondly by comparing the performance of metaheuristic search algorithms that were not previously utilised to solve this problem with the simulated annealing algorithm of their initial paper.

Chapter 6 focuses on applying metaheuristics to cryptanalysis. As part of this, so-called "nonlinear approximations" to S-boxes are evolved - these approximations could compromise the cipher's security against certain cryptanalytic techniques. However, the use of nonlinear approximations in cryptanalysis is complex, and not well understood. We therefore devote a significant proportion of this chapter to devising new cryptanalysis algorithms that can exploit nonlinear approximations, as well as statistical frameworks that can be used to calculate the complexity of the new techniques. We also discuss the application of the new techniques and evolved approximations to the cryptanalysis of various notable ciphers.

Chapter 7 is the concluding chapter, summarising the achievements of this research and discussing the extent to which our experiments have supported the hypothesis.

Appendix A contains the best evolved functions for small parameter sizes that resulted from the research in Chapter 4.

Appendix B is devoted to a discussion of a means to reduce the size of the search space when attempting to evolve S-boxes, and to its potential to lead (with future research) to an algorithm revealing whether two S-boxes are equivalent according to the concept of *affine equivalency*. While this did not result in significant improvements to the quality of the results obtained elsewhere in the chapter, it is nevertheless presented as an appendix due to the potential it may have if future research is applied to it.

Appendix C corrects a highly misleading error which occurred in several earlier papers on linear cryptanalysis of the cipher Serpent. The results of these papers are cited in Chapter 6, and compared to the work therein, hence the importance of addressing the error which seemed to invalidate them.

Chapter 2

A brief history of cryptography and cryptanalysis.

"Cryptography" is an umbrella term, encompassing several different problems in data security and their solutions. For example, digital signatures provide tamper resistance for documents transmitted over the Internet, both against malicious interference and random transmission errors, as well as confirmation that the document was written by the purported author. In this thesis, we will be applying metaheuristic search to ciphers - the oldest and best-known aspect of cryptography.

A cipher is a means of controlling access to data, by effecting a transformation on that data which can only be reversed by someone possessing the requisite piece of secret information - the "key". (Depending on the type of cipher, knowledge of the key may be necessary to effect the transformation as well as its reverse).

Effecting the above-mentioned transformation is known as "encryption", and reversing it, whether legitimately or otherwise, is known as "decryption". The algorithm taking the key and the data as input and outputting encrypted "ciphertext" data is the cipher, also referred to as the "cryptosystem". The inverse algorithm, converting the ciphertext back to the original "plaintext" data, typically has no name, or else the terms "cipher" and "cryptosystem" are assumed to encompass both algorithms.

A malicious third party may obtain copies of the encrypted data, and wish to illicitly reverse these transformations or obtain information on the key. Attacks on cryptosystems fall under the heading of "cryptanalysis", which, as its name suggests, is the analysis of cryptosystems (or any cryptographic algorithm) looking for weaknesses, regardless of

whether one intends to exploit these or to ensure that such weaknesses do not feature in future cipher designs!

Both umbrella terms themselves - cryptography and cryptanalysis - are referred to collectively by another umbrella term - "cryptology".

The history of cryptography is a long one. Very basic "pen and paper" methods are almost as old as writing itself; Mary, Queen of Scots corresponded in cipher during the reign of Queen Elizabeth I, and it was the cryptanalysis of these letters which revealed her involvement in a plot against Elizabeth and led to her execution [147, 218]. Notably, having broken the cipher, a forger working for Sir Francis Walsingham, Elizabeth's spymaster, was able to append an additional enciphered paragraph to one of the letters requesting the identities of the individuals who were to attempt the assassination. Believing that the secrecy resulting from the cipher could not have been compromised - and unaware that the means by which the enciphered messages were smuggled to/from Mary had been discovered and that said messages in both directions were being intercepted *en masse* - the leader of the conspiracy, Anthony Babington, sent an enciphered message back to Mary containing this information. His confidence in the security of his cipher prevented him from becoming suspicious when Mary, who did not need to know this particular information and would be unlikely to have even heard of the other conspirators, nevertheless requested their names!

Even further back, the Bible refers to a nonexistent place called "Sheshach". Biblical scholars eventually discovered that, by replacing the first letter in the Hebrew alphabet with the last letter, the second letter with the second-last letter, and so on, that "Sheshach" was in fact "Babel", or Babylon [147, 218]. Kahn [147] speculates that this was more likely to be wordplay by scribes than a serious attempt to disguise a place-name, however.

During the first half of the 20th Century, mechanical cipher machines came into existence; finding use protecting confidential information in businesses and the military. (These were typically too expensive to be within the reach of private citizens.) During the Second World War, one of these, Hagelin's M-209, was used by US Army soldiers to encrypt and decrypt messages in the field. The infamous Enigma cipher was the most well-known example of such a machine, requiring significant advances in computational technology and the development of electromechanical cryptanalysis machines at Bletchley Park to be broken!

The rise of the computer after World War 2 forced another major evolution in ci-

pher design. Computers were capable of cracking any practically-usable cipher that had gone before them, and to counter this it was necessary to construct ciphers which could themselves take advantage of the existence of the computer. This was also necessary to ensure that cryptography could be applied to sensitive documents stored on computers. Since expertise in cryptanalysis was typically concentrated within the highly-secretive intelligence agencies of governments, constructing a secure cipher for use by businesses and other non-government entities was no trivial task. Eventually, in the 1970s, a cipher based on IBM's "Lucifer" design was accepted as the new Data Encryption Standard (DES).

DES itself was not uncontroversial. The USA's National Security Agency had been involved in the adaptations made to the original Lucifer, and accusations were made that they had deliberately weakened the cipher to ensure that they could break it even if nobody else could. These accusations fell into two primary categories:

- The key, originally a 64-bit value, was now only 56 bits long. The number of possible values that would have to be tried to recover a key through exhaustive search was now reduced by a factor of 256, and it was widely alleged that the NSA had forced the reduction in key length to bring about a situation where it was within their computational capacity to crack DES in this way (but not within anybody else's).

- The NSA were also believed (accounts from IBM staff differ on this point) to have changed the design of certain parts of the cipher, the "S-boxes". These were lookup tables mapping six-bit inputs to four-bit output values. Statistical analysis of the S-boxes revealed that the values in the tables were not random, but did not find any way in which they introduced a weakness.

  It was, however, later discovered that the S-boxes had been designed to resist the attack known as "differential cryptanalysis" (described later in this document); an attack known to both IBM and the NSA at the time, but kept secret and not rediscovered until the early 1990s.

Other developments during the 1970s included the development of asymmetric "public key" ciphers such as RSA [208], in which the key value used to encrypt messages (the public key) was not the same as the value used to decrypt them (the private key). These were typically much slower than DES (or, indeed, the rival designs being developed), and would typically be used to securely transmit the key to a conventionally-enciphered full message. Nevertheless, the fact that a message could be securely encrypted, such that

only its intended recipient could read it, by someone with no knowledge of the private key or any other secret information, was thoroughly revolutionary.

With public key ciphers, the intention was that a potential recipient would publish their public key - some central server or servers would act as a repository of public keys and the email addresses of their owners, in effect a form of digital phone-book - and to send a message to someone, you would look up their public key and use it to encipher the message.

Another approach, *key-exchange*, was developed slightly earlier, in 1976. Let us illustrate this using an example, in which two parties want to agree on a secret key to use to communicate using a conventional cipher. Most examples of cryptographic protocols refer to a sender named Alice, and a recipient named Bob, trying to communicate without their messages being intercepted by an eavesdropper, Eve. In this case Alice and Bob have no means of securing their communications - any message sent in either direction could be intercepted by Eve. How can they still establish a shared secret key?

The Diffie-Hellman key exchange protocol [118] provided a solution to this problem using number theory:

1. Alice and Bob agree on a "public function" of the form $f(x) = g^x$ modulo P, where P is a large prime number and g is some integer less than P. It does not matter if the eavesdropper hears this and obtains the values of P and g.

2. Alice and Bob also each choose a private key. Each of these is another integer less than P. Let a denote Alice's key, and b denote Bob's. Neither party reveals their key.

3. Alice calculates $\alpha = f(a) = g^a \mod P$, and sends this to Bob. Bob sends $\beta = g^b \mod P$ to Alice. Even if Eve intercepts these communications, due to the difficulty of the so-called *discrete-logarithm problem* in number theory, she cannot deduce the values of a and b.

4. Alice receives β, and calculates $\beta^a \mod P = g^{ab} \mod P$. Bob calculates $\alpha^b = g^{ba} \mod P$. Since $g^{ba} = g^{ab}$, the two parties now have a shared secret number - and despite the amount of data she has been able to intercept, Eve cannot calculate it, because she cannot calculate either a or b!

(The Diffie-Hellman protocol is not absolutely perfect - it may be possible, for instance,

for a more active adversary, Mallory, to pose as Alice and trick Bob into establishing a shared key with him to communicate with. In the case of public-key cryptography, Alice can use what are known as "zero-knowledge proofs" to demonstrate that she knows the private key corresponding to the "Alice" public key without actually revealing anything about it to Bob - thus confirming her identity.)

Public key cryptosystems led to the concept of a "digital signature". Alice could use her private key in the same way as Bob's public key in the usual encryption procedure. She could then encrypt using Bob's public key, and send the message to Bob. After using his private key to decrypt the message, Bob would check the identity of the claimed sender (*alice@alice.com*), look up a directory of public keys, use Alice's public key in the decryption algorithm, and simultaneously

- complete decryption of the message, and
- confirm that Alice had written it.

The difficulty of digitally signing a full document would largely be overcome when *cryptographic hash functions* were later developed. A cryptographic hash function, such as MD5, SHA-1 or SHA-3, takes a document of arbitrary length and, using techniques similar to those used in symmetric ciphers, computes a short string of bits (precisely how "short" has varied - 128 bits for MD5, 160 for SHA-1, as many as 512 for SHA-3...) such that:

- Where h denotes the hash function, it should be computationally infeasible (i.e. impossible in practice) to deduce x from $h(x)$. This property is known as *preimage resistance.*
- Given some document x_1, it should be computationally infeasible to find some $x_2 \neq x_1$ such that $h(x_2) = h(x_1)$. This property is known as *second preimage resistance.*
- It should also be computationally infeasible to find any pair $(x_1, (x_2 \neq x_1))$ such that $h(x_1) = h(x_2)$. This property is known as *collision resistance.*

Cryptographic hash functions made it possible simply to sign a hash of the full document, instead of the document itself.

The Data Encryption Standard's 56-bit key value, whether or not it had been short enough to break through exhaustive search in the 1970s, most certainly was by the late

1990s. This forced a new, more open process to design DES's successor, and to subject it to cryptanalytic scrutiny using the new cryptanalytic techniques developed since DES's creation. Eventually, in 2001, the "Rijndael" cipher was selected as the new Advanced Encryption Standard [192].

After a survey in which the cryptanalysis techniques of differential and linear cryptanalysis are presented, along with a full survey of existing research in which metaheuristics have been utilised in cryptology, we present the results of our own investigation in this field.

Chapter 3

Literature review

3.1 Information on various types of block cipher, and a brief description of the Data Encryption Standard.

Several of the cryptanalytic techniques described in this document were originally designed to attack the Data Encryption Standard (DES). This cipher is of immense importance in the history of cryptology, partly due to its widespread usage by governments and businesses after its creation (and acceptance as a U.S. government standard) in the 1970s, and partly due to the amount of cryptologic research that has focused on it. As a widely used, publicly-known cipher declared secure by America's National Security Agency (NSA), it has been subjected to more scrutiny by cryptanalysts than any other cipher since the dawn of the computer age. We shall not describe it in detail here, but instead will provide a brief description of DES and various other ciphers/types of cipher, and refer the reader to the Federal Information Processing Standard document describing it [191] and an excellent online tutorial [124] for more details.

In brief, DES uses a 56-bit string as the key to encrypt a 64-bit block of data. The block is split into two 32-bit blocks; the "left-hand" and "right-hand" blocks. DES then repeats a procedure known as a "round" sixteen times, whereby

- A copy is made of the right-hand block.
- The copy is input to a function referred to as the "round function", along with 48 bits of the key.

- The particular subset of 48 bits varies from round to round, and is derived from the main key using an algorithm known as the "key schedule".
- It is essential that the round function should not be a linear function of the data input to it. The nonlinearity is provided by eight functions known as *S-boxes*. The cipher contains no other source of nonlinearity, and this is partly why the S-boxes have been subject to more intensive scrutiny than any other component of the DES.
- The left hand block is bitwise-XORed with the round function output.
- Unless this was the final round, the right and left-hand blocks are swapped around.

In addition, functions known as the "Initial Permutation" (IP) and "Final Permutation" (FP) are applied to the data before the first round and after the final round. However, these have no effect on the security of the cipher, and are usually ignored by cryptanalysts. (They seem to be intended to improve DES's efficiency by working around the limitations of 1970s hardware [211].)

3.1.1 Feistel ciphers

DES is a specific instance of a wider class of ciphers known as *Feistel ciphers*, which are in turn a subclass of the "block cipher" class - the class of symmetric ciphers operating on w-bit "blocks" of data.

A more generalised description of a Feistel cipher would be:

- The cipher acts on a $2n$-bit block of data, which is split up into two n-bit blocks. It uses a k-bit key, and has r rounds.
- In each round:
  - A copy is made of the right-hand block.
  - The copy is input to the "round function", along with a bitstring, the "round key" derived from the key by the "key schedule" algorithm. In most ciphers, all the round keys should be different.
  - As with the DES, it is essential that the round function should not be a linear function of the data input to it. S-boxes - functions equivalent to polynomials of high algebraic degree - are usually the chief source of nonlinearity in the round function. In some ciphers, the S-boxes differ between rounds.

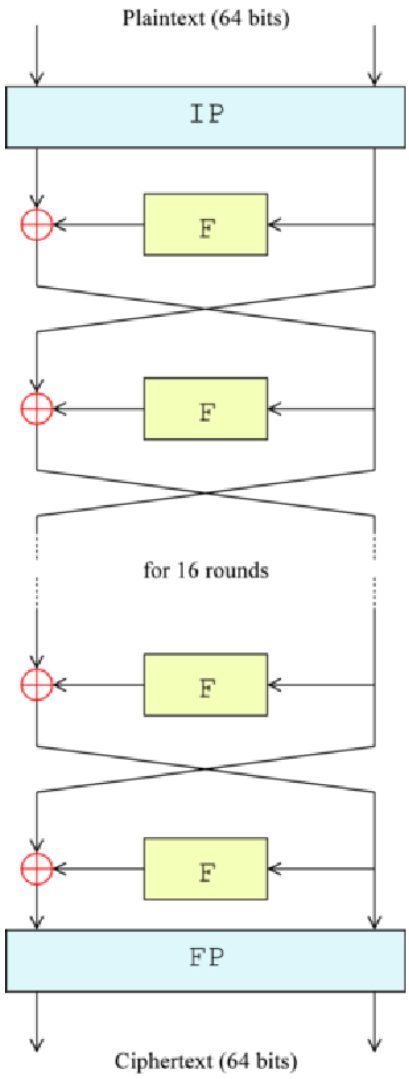

Figure 3.1: Diagram showing the basic "Feistel" structure of the Data Encryption Standard (DES). [231].

- The left hand block is bitwise-XORed with the round function output.
- Unless this was the final round, the right and left-hand blocks are swapped around.

Most Feistel ciphers do not have an IP or FP.

Feistel ciphers have the advantage that the same algorithm is used for encryption and decryption (although decryption requires a slightly different key schedule to reverse the order of the round keys). This is particularly useful for hardware implementations as it removes the need to create a second circuit/chip for the decryption process.

3.1.2 Other types of block cipher

Although the Feistel model has proved extremely successful, and is still used in designing ciphers today, it is not the only blueprint for block cipher design. There are also ciphers in which the data block is not split up at the start; instead the entire block and a round key are input to the round function - the output from which is input to the next round function along with another round key - and so on until the ciphertext is output. Examples of such ciphers include DES's successor, the Advanced Encryption Standard (AES) [192], the cipher for low-resource platforms known as "PRESENT" [40], and the simplified "Heys cipher" from Howard Heys's excellent tutorial on linear and differential cryptanalysis [140].

The Heys cipher is an example of a *substitution-permutation network* (SPN) [37]. In each round of such a cipher, all of the data in the block should interact with the key in some way, typically bitwise xor. The data is then input to several S-boxes in parallel. In all rounds except the last, a reordering (permutation) of the data bits occurs after this. In the last round, the data is typically xored with a final round key instead.

Two generalisations of such ciphers are described in the Encyclopedia of Cryptography and Security [37]:

1. Substitution-linear networks (SLNs), in which the permutation is replaced with some other linear function (AES is an example of an SLN), and:

2. Substitution-affine networks (SANs), a further generalisation in which the permutation is replaced with some affine function.

Although examples of the first such generalisation do exist (the Advanced Encryption Standard being one of them), the term "Substitution-linear network" has not entered

mainstream usage. Instead, the term "substitution-permutation network" has become generalised, and is now generally understood to refer to SLN ciphers as well.

Courtois and Pieprzyk [110] define "XSL-ciphers" as ciphers in which

X.) The entire data block is XORed with the round key at the start,

S.) after which each round function inputs the block to a layer of parallel S-boxes,

L.) applies some linear function to the result,

X.) and then XORs it with the round key again.

AES is an example of such a cipher in which the linear function applied in the final round is less complex than that applied in the other rounds. DES, while not an XSL cipher (its round function does not act on all the data), does have the same sort of round function, and the Heys cipher is similar to an XSL cipher except for the omission of the linear transformation from the final round.

A far more general category is the *iterative cipher* [36]. An iterative cipher is defined as a cipher in which the data and round keys are input to the same function (the round) several times in succession before being output. Minor differences (such as the values of certain constants) between the rounds are permitted, however the additional XOR and weaker diffusion layer in the final round mean that AES and the Heys cipher are not iterative ciphers. DES is probably the best-known example of an iterative cipher.

For most of this document, if we refer to a cipher without specifying which one we mean, unless otherwise stated it will, like the ciphers so far described, apply the same round function several times (with minor variations) to its data and subkeys.

3.1.3 Confusion and diffusion

Whatever its structure, to provide security for enciphered data, a cipher needs to provide high levels of *confusion* and *diffusion* [215].

Definition 3.1.1. Confusion complicates the relationship between the key and the ciphertext as far as possible. The S-boxes are the main source of confusion in most block ciphers, defining mappings that correspond to complicated polynomials that are not easy to approximate with simpler functions. (Functions mapping sets of bits to other sets of bits can always be expressed as collections of polynomials. It is necessary that these polynomials be of high degree.)

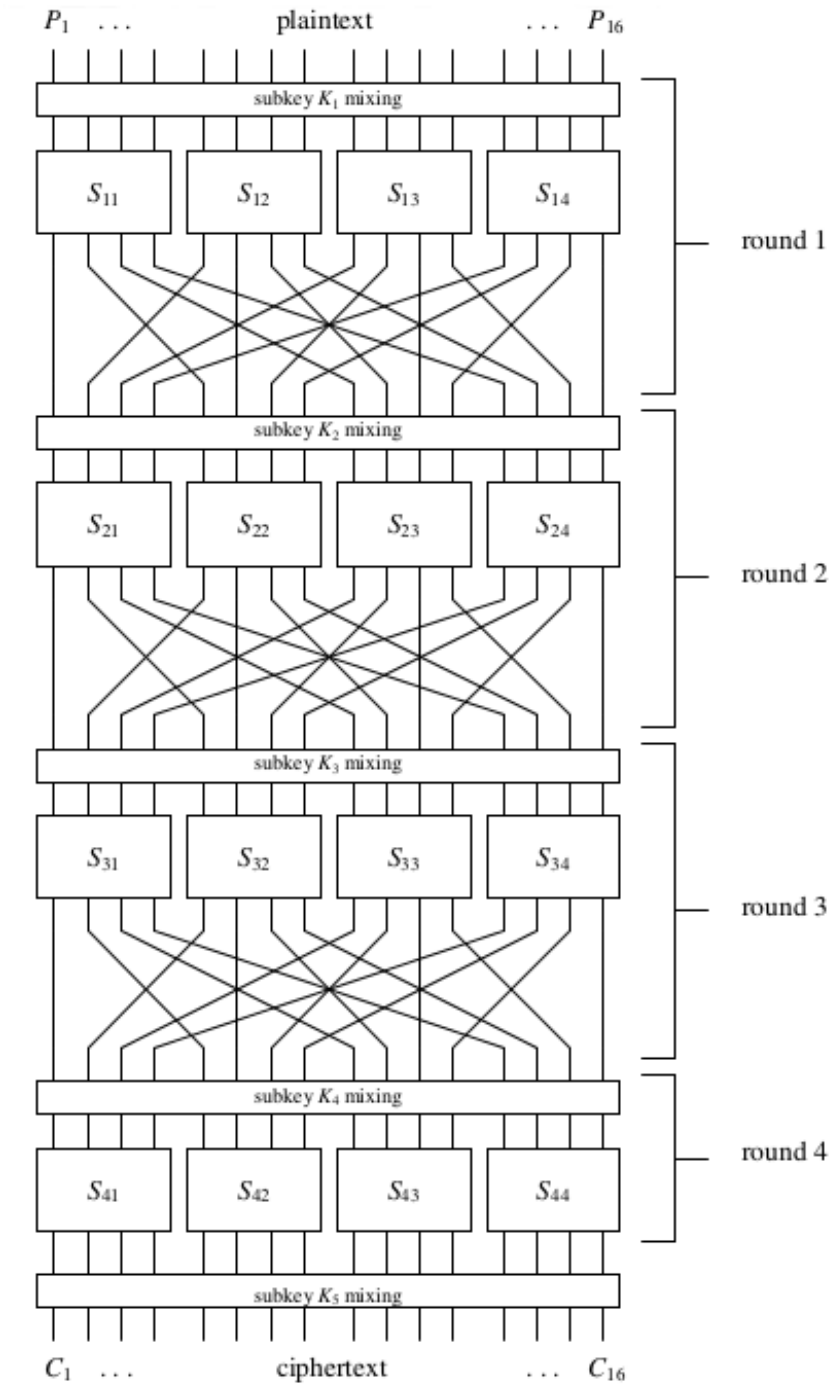

Figure 3.2: The Heys substitution-permutation network (SPN). [140].

Definition 3.1.2. Diffusion eliminates, as far as possible, any statistical properties from the ciphertext (In particular, it eliminates statistical properties of the original plaintext.). The higher the diffusion, the more data the cryptanalyst will need to observe any statistical properties that might aid in breaking the cipher. The permutations/linear functions/affine functions are the main source of diffusion in the cipher, and the S-boxes also contribute to this.

3.2 Linear cryptanalysis.

Linear cryptanalysis was first introduced by Mitsuru Matsui [179]. Where $\oplus$ indicates exclusive-or - notation we shall use throughout this thesis - the cryptanalyst attempts to find a linear equation $x_1 \oplus \ldots \oplus x_i = y_1 \oplus \ldots \oplus y_j$ in the input and output bits of some part of the cipher which holds true with probability sufficiently different to 0.5. "Sufficiently different" means that for a known-plaintext attack on a feasible number of known plaintexts, when the correct key (or part of it) is tried on all of these known plaintexts, the number of plaintexts for which the equation holds will deviate significantly from one half of the total.

Such an equation is known as a "linear approximation".

Definition 3.2.1. For a given linear approximation to part of a cipher, let p be the probability that it holds. We refer to $(p - 1/2)$ as the *bias* of the approximation. The higher the absolute value $|p-1/2|$ of the bias, the more useful the approximation is to the cryptanalyst.

The value $|p - 1/2|$ may also be referred to as the "absolute bias" or the "magnitude of bias". In a minor abuse of notation, when we refer to an approximation having high (or large, or significant) bias, we usually mean that the value of $|p - 1/2|$ is high.

Matsui refined his attack in a 1994 paper [180], in which he used it to attack the Data Encryption Standard. The refined attack could break the full 16-round DES with an estimated 2^{43} time complexity (that is, in an amount of time equivalent to that required for 2^{43} DES encryptions) and with an 85% probability of success. This time complexity was the best achieved by any attack on DES at the time, and has not subsequently been improved upon!

However, it also required 2^{43} known plaintexts; and the difficulty of obtaining these, as well as the resources required to store them, meant that a brute force attack was in practice

more feasible. This remained the case despite later work by various authors suggesting that the attack could succeed with less resources:

- Nyberg [198], and Harpes, Kramer and Massey [130] carried out analyses which showed that some of Matsui's assumptions had been overly pessimistic, and that the true complexity of the attack would be better than he had predicted (although new complexity estimates were not provided.)

- Junod [146] provided evidence that the time required by Matsui's attack was in fact equivalent only to that required to perform 2^{41} DES encryptions.

- Biryukov et al. [38], conducting research into the use of several linear approximations simultaneously (Matsui's attack had used two) conjectured that the attack's data requirements could be reduced to 2^{41} using a particular set of 108 approximations. (They do not seem to have successfully implemented this attack, however.) They also conjectured that, although new techniques would be needed, using approximately 10,000 approximations simultaneously would reduce the number of known plaintexts needed to 2^{36}.

  It should, however, be noted that there is some controversy surrounding this particular piece of research [188, 189].

Any new cipher must be shown to be resilient to linear cryptanalysis before it can be considered for use. There are now block cipher design strategies, in particular Joan Daemen and Vincent Rijmen's *wide trail* strategy [112] and Serge Vaudenay's *decorrelation theory* [225, 226, 227, 228], which can be used to design block ciphers resilient against linear cryptanalysis, differential cryptanalysis ([33, 32, 34] - the subject of Section 3.2) and their variants. The first of these was in fact used to design the Advanced Encryption Standard [113].

3.2.1 The attack.

Overview.

In an attack on a cipher, linear cryptanalysis is typically used in one of two ways. Both of these require a large volume of known (plaintext, ciphertext) pairs:

The 1R attack. Let r denote the number of rounds in the cipher. A linear approximation is found for the first $(r-1)$ rounds. The cryptanalyst looks at how the output bits of this linear equation are input into the final round, and partially deciphers it with her guessed values for the relevant key bits. Only for the correct guess at these bits - the "target partial subkey" (TPS) - should the approximation hold true as often as expected.

The reason why the cryptanalyst is able to partially decipher the final round using only some of the key is that for several block ciphers, a given input bit to the round function will not affect all of the output bits. (For instance, in DES, no round function input bit affects more than eight of the 32 output bits.) Furthermore, the complete set of input bits affecting certain sets of output bits is usually relatively small, the output bits being partitioned up and each partition assigned to a small subset of the input bits. The cryptanalyst therefore only needs to use the key bits acting on that subset of the input bits.

The number of key bits that should comprise the TPS is something the cryptanalyst will need to decide for herself. It must be small enough for an exhaustive search over all possible TPS values to be possible, but the cryptanalyst needs to consider how the remaining key bits will be obtained. Exhaustive search? More linear attacks with a different target partial subkey? This is something that will probably depend on various different factors.

As well as the TPS, one more bit of key information can be obtained using this attack; although this will take the form of a linear equation relating some of the key bits. We will explain later how to identify these key bits, but basically whether they xor to 1 or 0 can be determined by whether the linear approximation held or did not hold for most of the plaintexts - for the rest of the attack this does not matter as long as they do one or the other with the expected bias.

The attack algorithm for this final key bit was described by Matsui as "Algorithm 1". Somewhat counterintuitively, the algorithm for the main attack on the bulk of the key bits, which precedes the attack on that bit, is "Algorithm 2"!

The 2R attack. Matsui's second paper defined the "2R" attack; which he used in his attack on DES. The cryptanalyst finds a linear approximation for all of the cipher's rounds except the first and the last. As before, she uses candidate TPS values to

decipher the final round output bits affected by the linear approximation's output bits. However, the TPS bits used in this no longer comprise the whole TPS - for another subset of key bits, she encrypts some of the bits of each known plaintext to obtain the linear approximation's input bits. She then checks to see whether the linear approximation holds.

As before, the correct TPS is expected to be the only one such that the approximation holds with the expected bias, and one more bit of key information can be obtained depending on whether it held or did not hold for most of the trials.

We stop to note that differential cryptanalysis, which is in many ways extremely similar to linear cryptanalysis, seems to be more flexible in terms of which rounds can and cannot be covered by their equivalent to the linear approximation. For instance, Biham and Shamir [34] attacked DES using a differential attack in which two rounds at the end were not covered, and in which the first round was intended to cause the input to the remaining rounds to have the properties required.

Deriving the approximation.

Let us assume that we are trying to cryptanalyse a Feistel cipher like DES, or an SPN. Typically the only parts of these ciphers which cannot be expressed as linear Boolean formulae are the S-boxes.

Therefore, for each S-box and every possible subset of the input and output bits to it, we calculate using all possible inputs whether the xor of that subset of the input bits is equal to the xor of that subset of the output bits. From this, we calculate the probability bias of every possible linear approximation to the S-box.

We assign to each S-box a *linear approximation table*, in which we record these biases. An example is shown below, in Figure 3.3.

The next step is to link these linear approximations together to form an overall approximation of the entire cipher except for the last (and maybe the first) round. Finding the best such approximation, or even a "good enough" approximation is not a trivial task. Nor is it obvious how best to do this, and we intend at some point in the future to find out if metaheuristic search techniques can be applied here. One currently-employed tactic is to find a good approximation for a small number of rounds such that the subset of input bits is the same as the subset of output bits. This allows the approximation to be iterated several times to approximate the full set of $r-1$ or $r-2$ rounds.

		Output Sum															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Input Sum	0	+8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	−2	−2	0	0	−2	+6	+2	+2	0	0	+2	+2	0	0
	2	0	0	−2	−2	0	0	−2	−2	0	0	+2	+2	0	0	−6	+2
	3	0	0	0	0	0	0	0	0	+2	−6	−2	−2	+2	+2	−2	−2
	4	0	+2	0	−2	−2	−4	−2	0	0	−2	0	+2	+2	−4	+2	0
	5	0	−2	−2	0	−2	0	+4	+2	−2	0	−4	+2	0	−2	−2	0
	6	0	+2	−2	+4	+2	0	0	+2	0	−2	+2	+4	−2	0	0	−2
	7	0	−2	0	+2	+2	−4	+2	0	−2	0	+2	0	+4	+2	0	+2
	8	0	0	0	0	0	0	0	0	−2	+2	+2	−2	+2	−2	−2	−6
	9	0	0	−2	−2	0	0	−2	−2	−4	0	−2	+2	0	+4	+2	−2
	A	0	+4	−2	+2	−4	0	+2	−2	+2	+2	0	0	+2	+2	0	0
	B	0	+4	0	−4	+4	0	+4	0	0	0	0	0	0	0	0	0
	C	0	−2	+4	−2	−2	0	+2	0	+2	0	+2	+4	0	+2	0	−2
	D	0	+2	+2	0	−2	+4	0	+2	−4	−2	+2	0	+2	0	0	+2
	E	0	+2	+2	0	−2	−4	0	+2	−2	0	0	−2	−4	+2	−2	0
	F	0	−2	−4	−2	−2	0	+2	0	0	−2	+4	−2	−2	0	+2	0

Figure 3.3: The linear approximation table for the example S-box in the Heys toy cipher [140]

Building up the full approximation. We will discuss joining up the individual approximations later, but let us first look at how we build up the set thereof. Consider an input bit that forms part of an $(r-1)$-round linear approximation to the cipher. We have to incorporate a linear approximation to the first round S-box it enters, and that input approximation can only involve the input bits from our overall approximation that actually enter it.

For the output bits involved in that linear approximation, we trace each one through the cipher until it enters another S-box. We will need to incorporate a linear approximation to that S-box as well. And, once again, the input bits of this approximation must be the ones we traced to that S-box... Eventually, a set of branching paths are traced through the cipher passing through various S-boxes along the way until they get to the approximation's output bits.

(Note that joining linear approximations together is more complicated in the case of a Feistel cipher. Whenever a path forks, you only follow the bits down *one* of the forks, and must choose which. In addition, you might have built up some of the approximation by starting from the last-but-one round and working backwards, or by working outwards from one of the rounds in the middle... This makes such diagrams for Feistel ciphers like DES much harder to follow. We may appear to "skip" rounds that did not need to be approximated, and round input bits resulting from working backwards can seem to appear

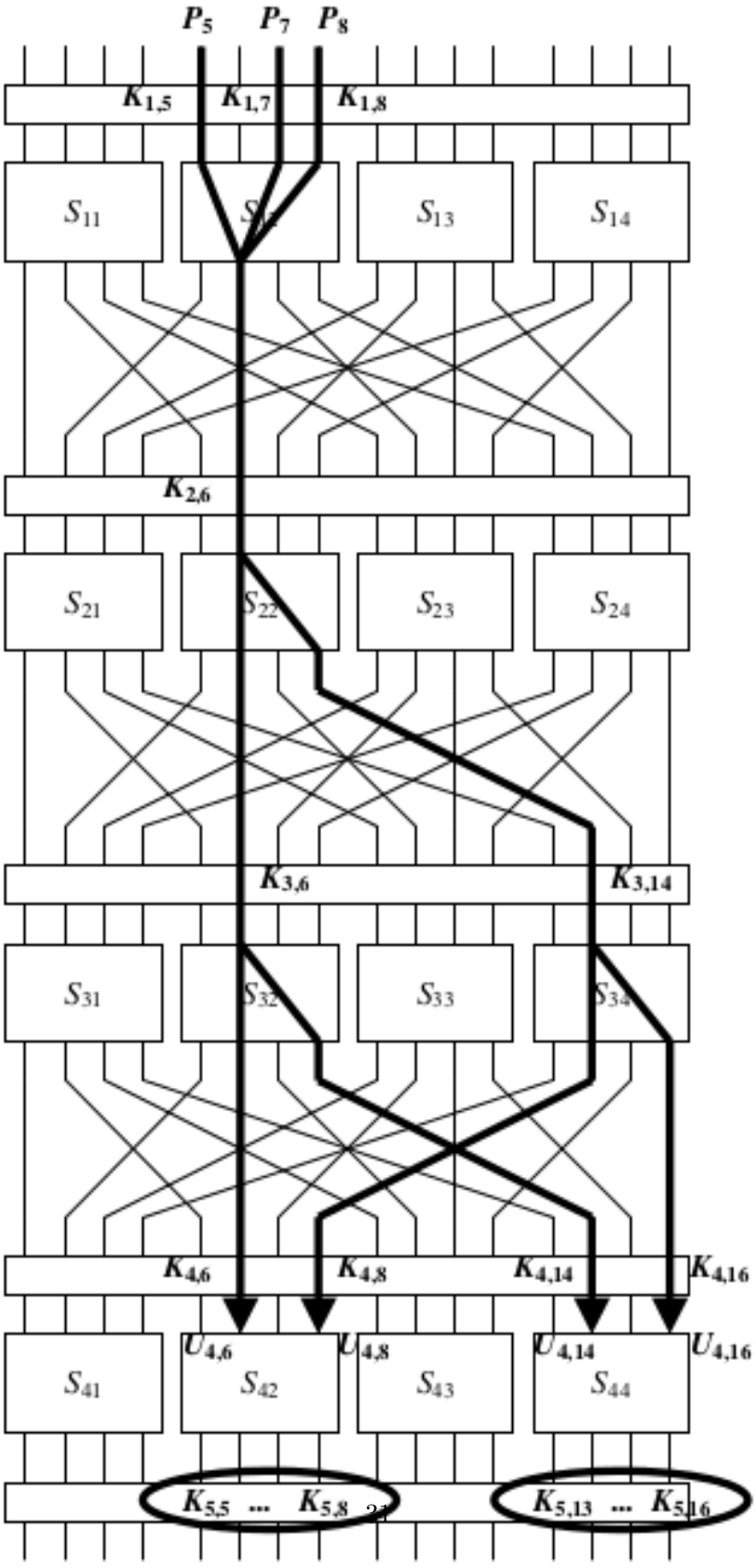

Figure 3.4: A linear approximation to the first three rounds of the Heys SPN.

out of nowhere if you try to trace the approximation from the start of the cipher.)

When choosing a linear approximation for each S-box, there are two desirable properties which, unfortunately, may conflict with each other. Both of them result from the same fact; that the bias of the overall approximation is determined by multiplying together the biases of the individual approximations.

Lemma 3.2.2. *For each value* $(1 \leq i \leq n)$*, let* X_i *be a random variable, independent of* X_j *for all* $j \neq i$*, such that:*

$$P(X_i = 0) = p_i$$
$$P(X_i = 1) = (1 - p_i)$$

Then $P(X_1 \oplus X_2 \oplus \ldots X_n = 0)$ *is:*

$$\frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2).$$

This lemma is known as the Piling-Up Lemma [179].

We use the Piling-Up Lemma in estimating the overall bias. Each X_i corresponds to a linear approximation, taking the value 1 if it holds and 0 if it does not. This looks at first sight like an exact calculation; however the various X_i are not in fact independent of each other. Despite this, the values given by the Piling-up Lemma have been observed to work well in practice, and do in fact slightly underestimate the probability p that the approximation holds (Empirical evidence is given by experiments on DES and other ciphers [146] [97]).

By examining the Piling-up Lemma, we see that it is desirable for each individual S-box approximation to have a bias with a high absolute value, and for there to be as few individual approximations as possible. In general, though, for a given S-box the approximation with the most significant bias may not be the one that has the fewest output bits, so choosing it may push up the number of S-boxes we will have to approximate in the next round. This was certainly the case for DES, where the design team were particularly careful to avoid good linear approximations with only one output bit [101]. Similarly, the S-boxes of the cipher Serpent [5] were designed so that any linear approximation with one input and one output bit could not have a bias exceeding 1/8 in magnitude; whereas the

upper bound for absolute biases of other linear approximations was 1/4.

Launching the attack.

Having found a good approximation, we next do whatever is necessary to obtain our known plaintext/ciphertext pairs. An integral equation relating the number of pairs N required to the desired success rate (and certain other aspects of the attack) is given by Selçuk [213]. This is used to determine c such that $N = c/|p - 1/2|^2$ is necessary to achieve the desired success rate.

If our TPS has t bits, we will also need enough memory to hold 2^t integer values, one for each possible value of the TPS. The integer variable corresponding to TPS value T_i is referred to as the *counter* for T_i. At the start of the attack, we initialise all these counters to zero.

Once we have obtained our pairs, for each pair and for each TPS guess T_i we decrypt the relevant parts of the ciphertext (and encrypt the relevant parts of the plaintext.), and check to see if our linear approximation holds for the thus-exposed bits. If it does, we add 1 to the counter for T_i.

(For most ciphers, a highly optimised algorithm exists that allows a "short cut" to be taken in calculating the counter values, instead of testing all (pair, TPS guess) combinations. This is quite complicated, and the reader is referred to Chapter 6 for more information.)

We expect that the correct guess will be the only one for which the calculated probability bias is seen to occur. For the others, the bias observed should usually be extremely small. However, in practice other values will also produce significant biases, although not as significant as the correct one.

There are various possible reasons for this, stemming in part from the fact that the number of possible key guesses is often high enough for a minority of the incorrect guesses to have a bias several standard deviations away from the low expectation, but also from the fact that the biases for certain incorrect key guesses are not statistically independent of the bias resulting from the correct guess.

What, then, should we do about these other significant biases? If we have been able to acquire a large number of known plaintexts, the bias observed for the correct TPS may be the largest by a noticeable margin, allowing us to simply ignore them. Perhaps, though, we cannot be sure that the highest bias observed corresponds to the correct TPS. If so,

we may want to assume that the largest bias corresponds to the correct value at first, but if all possible values for the remaining key bits do not result in any successful decryptions, switch to the value with the next-highest bias - and so on, up to some predefined limit on the number of TPS candidates tried. This approach is referred to as *key ranking*.

One question that has not hitherto been asked is whether these biases, which seem similar to local optima, might in some way be useful in applying nature-inspired search techniques like hill-climbing to the attack. It seems unlikely that hill-climbing can be directly employed, but it may be possible, by analysing how these biases occurred, to obtain some useful information. We hope to be able to investigate this further at some point in the future.

The other bit of key information. As stated earlier, one more bit of key information can be obtained depending on whether the linear approximation held, or failed to hold, with the bias predicted. However, we have not yet really explained this.

Let us consider a simplified approximation, in which the linear approximation is traced through two 4×4-bit S-boxes. Like DES, we will assume that the data bits are xored with key bits before they enter the S-boxes. Let the first S-box be denoted S_1, and assume that the only plaintext bit P_1 entering it can be traced to input bit x_2 of S_1. Now, before that bit actually does enter the S-box, it is xored with some key bit - call it k_1.

If our linear approximation involves only one output bit, say y_4, then our linear approximation is $x_2 = y_4$ and has probability p_1. Since $x_2 = P_1 \oplus k_1$, $P_1 \oplus k_1 = y_4$ with probability p_1.

Let us trace this output bit to the second S-box, S_2. Assume it enters the S-box as input bit x_3. Again, it would be xored with a key bit - call it k_2 - before entering it. Let the output bits this time be y_2 and y_4. We have the linear approximation $x_3 = y_2 \oplus y_4$, holding with probability p_2.

We now need to link the linear approximations together. We make a slight change to our notation so that the input and output bits have a superscripted number identifying their corresponding S-box:

1. $x_2^1 = y_4^1$, which we translate to $P_1 \oplus k_1 = y_4^1$.
2. $x_3^2 = y_2^2 \oplus y_4^2$, which we translate to $y_4^1 \oplus k_2 = y_2^2 \oplus y_4^2$. Rearranging this gives us $y_4^1 = k_2 \oplus y_2^2 \oplus y_4^2$.

By substituting for y_4^1, we are now able to link the two approximations together and obtain $P_1 \oplus k_1 = k_2 \oplus y_2^2 \oplus y_4^2$. For our simplified example, we will assume that the cipher only has three rounds, and so we approximate only these two. The overall approximation has probability $p = p_1 \times p_2$.

Let us rearrange that equation. With probability p, $P_1 \oplus y_2^2 \oplus y_4^2 = k_1 \oplus k_2$. Hence, if for the TPS value we accept as correct $P_1 \oplus y_2^2 \oplus y_4^2 = 0$ for $\approx p \times N$ pairs, we assume that the xor of those two key bits is zero - and vice versa.

3.3 Differential cryptanalysis.

Differential cryptanalysis, published in 1990 by Biham and Shamir [32, 33], was the first notable cryptanalysis technique for Information Age ciphers to be discovered outside the world's intelligence agencies. It was the first technique to allow an attack on DES faster than exhaustive search [34]; although as with linear cryptanalysis the number of chosen plaintext/ciphertext pairs required meant that exhaustive search was still more feasible in practice.

(Note the reference to "chosen" pairs instead of "known" - the cryptanalyst must, for *this* attack, ensure not only that there are enough plaintext/ciphertext pairs, but also that they satisfy certain properties...)

Differential cryptanalysis is very similar to linear cryptanalysis; a "differential characteristic" is built up to cover some of the rounds by calculating individual characteristics for various S-boxes and then joining these together. In this case, we are trying to find a situation where, for some pair of plaintexts (P_i, P_j) such that

in a 1R attack $P_i \oplus P_j =$ a particular "input difference" value ΔX.

in a 2R attack The partial encryption Q_i of P_i, $\oplus$ the partial encryption Q_j of P_j, $= \Delta X$. $P_i \oplus P_j$ is in the set of values such that this is possible for at least one key.

the partial decryptions (D_i, D_j) of the corresponding ciphertexts (C_i, C_j) will take a particular "output difference" value, $D_i \oplus D_j = \Delta Y$ with sufficiently high probability.

To build up the corresponding differential characteristic, we join together the individual S-box characteristics into characteristics for individual rounds (as we did for linear cryptanalysis), and then join together the round characteristics by letting the output difference from one round be the input difference into the next. For a particular set of key

		Output Difference															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Input Difference	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	2	0	0	0	2	0	2	4	0	4	2	0	0
	2	0	0	0	2	0	6	2	2	0	2	0	0	0	0	2	0
	3	0	0	2	0	2	0	0	0	0	4	2	0	2	0	0	4
	4	0	0	0	2	0	0	6	0	0	2	0	4	2	0	0	0
	5	0	4	0	0	0	2	2	0	0	0	4	0	2	0	0	2
	6	0	0	0	4	0	4	0	0	0	0	0	0	2	2	2	2
	7	0	0	2	2	2	0	2	0	0	2	2	0	0	0	0	4
	8	0	0	0	0	0	0	2	2	0	0	0	4	0	4	2	2
	9	0	2	0	0	2	0	0	4	2	0	2	2	2	0	0	0
	A	0	2	2	0	0	0	0	0	6	0	0	2	0	0	4	0
	B	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2
	C	0	2	0	0	2	2	2	0	0	0	0	2	0	6	0	0
	D	0	4	0	0	0	0	0	4	2	0	2	0	2	0	2	0
	E	0	0	2	4	2	0	0	0	6	0	0	0	0	0	2	0
	F	0	2	0	0	6	0	0	0	0	4	0	2	0	0	2	0

Figure 3.5: Difference distribution table (DDT) for the 4×4 S-box used in the Heys cipher [140].

bits (the TPS), we partially decrypt the cipher and, for each possible value of these bits, maintain a count of how many times the correct output difference occurred.

Definition 3.3.1. For each S-box, we construct a table. Each column corresponds to a given output difference, each row to a given input difference. Each entry in the table is the number of pairs of S-box inputs with input difference corresponding to the row that map to pairs with the column's output difference.

We call these tables *difference distribution tables* (see for example Figure 3.5.)

Because of the need to have pairs of plaintexts with the given input difference, or with input differences in a particular set, differential cryptanalysis is a *chosen-plaintext* attack, unlike linear cryptanalysis which was a *known-plaintext* attack. This makes it harder to carry out, as the cryptanalyst has to be able to obtain ciphertexts corresponding to specified pairs of plaintexts instead of just random plaintexts. (In fact, variants of the attack employ even more complicated sets of plaintexts related in specific ways, such as quartets $(P_1, P_2 = P_1 \oplus \Delta X_1, P_3 = P_1 \oplus \Delta X_2, P_4 = P_2 \oplus \Delta X_2)$.)

As stated, by joining together the differential characteristics, we obtain a *differential* for the rounds in which they feature, $(\Delta X, \Delta Y)$. Let y denote the number of output bits

affected by ΔY. We are trying to choose and link up differential characteristics so that ΔY will be the output difference for those y bits significantly more often than $1/2^y$ of the time, given that the input difference is ΔX.

An important factor in choosing and linking together differential characteristics is that, for ciphers like DES where the key interacts with the data only by being xored with it before entering the S-box, it will have absolutely no effect on the characteristic! Think about an S-box with six input bits and four output bits. Let us assume that we are working with input difference 010000 (in other words, inputting pairs that differ only in bit x_2) and output difference y_1.

Where the superscripted number identifies whether a data value corresponds to the first or second input in the pair, input difference 010000 means that $x_2^1 \oplus x_2^2 = 1$, and $x_i^1 \oplus x_i^2 = 0$ for all other i. So, where the corresponding data bits are d_2^1 and d_2^2, and the corresponding key bit is k_2, $(d_2^1 \oplus k_2) \oplus (d_2^2 \oplus k_2) = 1$, which reduces to $(d_2^1 \oplus d_2^2) = 1$ - the key bit has no effect!

In general, differential cryptanalysis has not been as limited as linear cryptanalysis in terms of the rounds that can be covered by characteristics. A given attack, designed for maximum effectiveness, may cover only the first $r - 1$ rounds [140], it may cover the first $r - 3$ rounds [32], it may cover rounds 2 to $r - 2$ and use the first round differently [34] ... All this varies depending on the cipher and any tweaks made by the cryptanalyst to optimise the attack for it, although we will say that substitution-permutation networks, since their round function acts on the whole data block, will probably require more rounds to be covered as the diffusion throughout the remaining rounds will be faster.

The overall differential characteristic is considered not to be the differential $(\Delta X, \Delta Y)$, but the set of input and output differences for each of the various rounds of the cipher, accompanied for each round by the set of S-boxes involved and the input/output difference we chose for each. In practice, the characteristic is the set of individual characteristics that we used to build the differential. Even if the differences specified by the differential characteristic do not occur, the differential may still occur (that is, the output difference may be ΔY for input difference ΔX), so the probability of the overall characteristic is a lower bound for the probability of the differential.

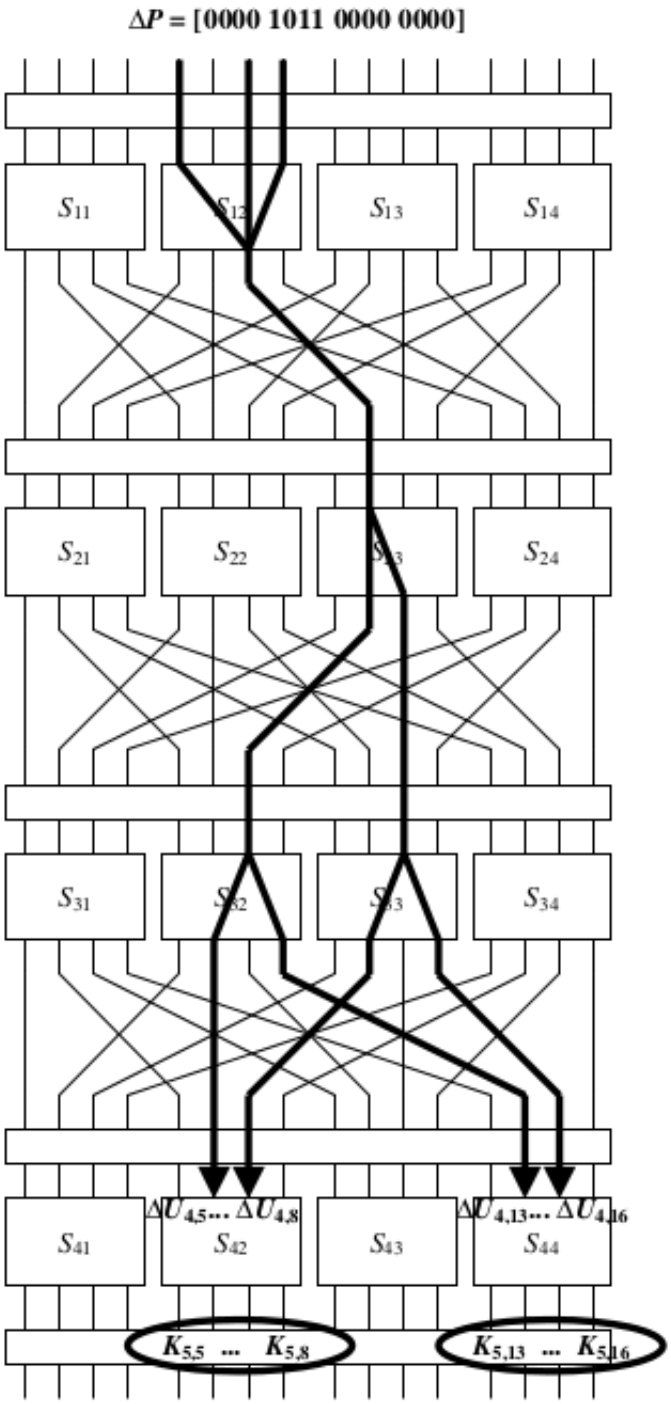

Figure 3.6: A differential characteristic linked across several rounds of the Heys SPN.

3.3.1 The attack.

Obtaining ciphertexts.

We begin by obtaining the ciphertexts corresponding to our chosen plaintexts, by whatever means are necessary.

Definition 3.3.2. Let $((P_i, C_i), (P_j, C_j))$ be a chosen plaintext pair. Let (Q_i, Q_j) be the partial encryptions of P_i and P_j, and (D_i, D_j) the partial decryptions of (C_i, C_j). If $Q_i \oplus Q_j = \Delta X$ and $D_i \oplus D_j = \Delta Y$, then this is a *right pair*. If not, it is a *wrong pair*.

How many chosen plaintexts do we need? Assume that we are counting the number of times the expected output difference occurs for each TPS candidate. Let p denote the probability that a given pair involved in the attack is a right pair. For the correct TPS, we need the expected output difference to occur at least once, so we need at least one right pair to be present. We expect one right pair to be present in $1/p$ pairs, so $2/p$ plaintexts would be a reasonable lower bound to start with. (Not all differential attacks work like this, though - the "memoryless" variant exploits the quartet structure to require only $1/p$ plaintexts.)

Deriving a better figure is rather complex. We will explain it in more detail after we have discussed some "preprocessing" done on the (P_i, C_i) pairs before the main attack.

Discarded pairs Before the main attack, we can discard certain pairs which cannot possibly be right pairs. Consider the way in which our differential characteristic travels through various S-boxes of the cipher, and in particular the ones it does *not* pass through in the final round covered by the characteristic. If the characteristic holds, then since there was no difference in the input to these S-boxes, there should be no difference in output either. We trace this effect through the final, uncovered round(s) of the cipher to identify ciphertext bits that should be identical.

Hence any chosen plaintext pair in which some of these bits differ in the corresponding ciphertexts should be immediately discarded, as the differential characteristic could not have held. We cannot check the plaintext for anything like this, though, as we chose the plaintexts so that they only differed in the right places.

It may be possible to discard more pairs based on the S-boxes which *are* involved in the final round of the characteristic. For the input differences specified, certain output differences occur with probability zero - these are called *impossible differentials*. If it

is possible to spot that these have occurred without a partial decryption (and this is sometimes, but not always, the case) we can discard the pairs in which they occurred. Again, we cannot do this in the first round, as we would have chosen the plaintexts to avoid such impossible differentials.

These procedures will discard pairs which are certain to be wrong pairs. Shamir and Biham also describe [32] a method of discarding pairs which have high probability of being wrong pairs. This seems in practice not to have discarded too many right pairs, and to have helped improve the effectiveness of the attack.

For a given pair, look at the actual output differences that have occurred for the S-boxes involved in the final round of the characteristic. For the predicted input differences for these S-boxes, and the observed output differences, we multiply together the corresponding values in their difference distribution tables to obtain what is known as the *weight* of the pair. Shamir and Biham state (without proof) [32] that a right pair will typically have a higher weight than a wrong pair, and that for the attack on DES presented in that paper eliminating all pairs with weight below 8192 "discarded about 97% of the wrong pairs" while leaving "almost all of the right pairs".

There may be ways to discard more pairs for a given cipher, but in any case we discard all the pairs we can and are left with a set which should contain a higher concentration of right pairs.

Suggested subkeys.

Definition 3.3.3. We say that a pair of chosen plaintexts "suggests" a given value for the TPS if the correct differences $(\Delta X, \Delta Y)$ occur for that pair and that TPS candidate.

In working out how many chosen plaintexts we will need, the average (mean) number of TPSes suggested by each pair is also important.

As an example of how to calculate this, let us make the simplifying assumption that this is a 1R attack and our characteristic covers all rounds except the final round. Consider a final-round S-box that is affected by our TPS, and the expected input difference to it. We refer to the subset of the TPS bits corresponding to this S-box as the TPSS (target partial sub-subkey). For every possible output difference from that S-box, let us total up the number of pairs with the expected input difference that can give rise to it, then divide by the total number of possible output differences.

Lemma 3.3.4. *For conventional S-boxes such as those used in DES or AES, this figure is* $2^{|input\ bits|-|output\ bits|}$. *We do not provide a proof here, but with knowledge of the properties of difference distribution tables it is easy to prove.*

This value is the number of TPSSes suggested by the S-box. This is because, for a randomly chosen output difference in the ciphertext, it is the average number of keys that could have turned the input pair into one of the ones that could result in that difference.

We multiply the figures we have obtained for each S-box, and obtain a, the average number of target partial subkeys suggested by each pair.

The signal to noise ratio.

Definition 3.3.5. Let p be the probability of the differential characteristic used in the attack, a be the average number of TPS candidates suggested by each pair, b be the ratio |non-discarded pairs|/|total number of pairs|, and m be the number of bits in the TPS.

The *signal to noise ratio*, denoted S/N, is

$$\left(\frac{2^m \times p}{a \times b}\right).$$

Using S/N, which we now have enough information to work out, we can obtain a better estimate for the number of pairs we need. Biham and Shamir state [32] that experiments indicate an S/N between 1 and 2 requires between $40/p$ and $80/p$ chosen plaintexts, and that the higher the signal-to-noise ratio, the less right pairs we need. S/N is in fact a rearrangement of an equation for (number of right pairs present)/(number of times an average TPS gives the correct output difference), and hence a lower bound for (number of times the right key is counted)/(number of times an average key is counted.) Hence, if the signal-to-noise ratio is too far below 1, the attack will fail.

(This does not necessarily mean that variant attacks like *impossible differential cryptanalysis* will fail, though.)

After we have obtained the pairs.

Counting on every possible TPS. The most basic way to carry out a differential cryptanalytic attack is, for each pair of known plaintexts, to partially encrypt them, and decrypt their corresponding ciphertexts, with every possible TPS. We allocate an integer

variable - a "counter" - to each TPS candidate, initialise it to 0 at the start, and increment it by 1 whenever the expected output difference occurs for its corresponding subkey.

We either accept the TPS with the largest count as the correct one, or (if we are not so sure that it has a commanding lead) start off by assuming it to be correct, but switch to the one with the next highest count if it turns out not to be, and so on (up to some predetermined limit on the number of TPS values tried). As we did with linear cryptanalysis, we use the term "key ranking" for this approach.

Note that we are interested in the largest counts, not the ones that deviated the most from |plaintext pairs|/2. This is an important difference between differential and linear cryptanalysis.

The clique method. The clique method was introduced by Biham and Shamir to deal with TPSes so large that the amount of memory required to assign a counter to each one would not be feasible. It is, however, only viable if a relatively small number of pairs are being analysed (though since a large TPS will result in a large signal to noise ratio, that may not be an unreasonable assumption.)

Let us associate with each pair some form of data structure to keep track of the TPS values it suggests. If a is low enough, a linked list would seem to be a good way to do this; however Biham and Shamir did not make this assumption, and constructed an alternate data structure using less memory but which would result in false positives for some candidate keys. In practice this does not seem to have been a problem for them, though.

For each S-box affected by the TPS bits, we allocate $2^{(\text{TPS bits affecting that S-box})}$ bits of storage to each pair. Typically this will be $2^{\text{input bits}}$ per S-box, one for each possible sub-TPS affecting it. We initialise them to zero.

Whenever a particular TPS is suggested by the pair, for the sub-TPS corresponding to each S-box we set its bit to 1. So, by concatenating sub-TPSes whose corresponding bits have been set to 1, the idea is that we thus reconstruct a suggested key. This is where the false positives issue becomes relevant - if two keys are suggested, and if five S-boxes are involved, we may have to set two bits to 1 for every S-box. This means that we have in practice recorded up to $2^{|\text{involved S-boxes}|} = 32$ keys as suggested when only two were! The reason this did not in fact cause problems may have been that no individual false positive of this sort was suggested for particularly many pairs, but this is not stated explicitly.

We then need to find out which key is recorded as having been suggested by the most

pairs. Again, a linked list of objects (each object being a pair and its associated data on suggested keys) would seem to be a useful data structure to use.

For the first pair in our list, we iterate through the list until we find another pair such that they both suggest a common key. We then continue to iterate through the list looking for other pairs that suggest this key. We keep track of how many suggested it, then look for pairs that have different keys in common with the first... Eventually, we reach the bottom of the list, and start looking for pairs which suggest keys that the second pair also suggested.

The more pairs that suggest a particular TPS candidate, the more likely we consider it to be the correct one.

When the TPS is too long for the first method, and there are too many pairs for the clique method. In this case, we will have to use the conventional method, except that we do not attack the full TPS because we are not counting on all the S-boxes. In other words, we keep counters for every possible value of a smaller TPS, defined by removing bits from the actual TPS corresponding to S-boxes we decided not to count on - call them *redundant* S-boxes.

We will need a particularly high number of pairs for this to work, as the reduced TPS size will reduce S/N. We can mitigate this somewhat by checking, for the redundant S-boxes, whether impossible input/output difference combinations have occurred and using this fact to discard more pairs, however.

After recovering the sub-TPS, we may be able to use the redundant S-boxes in a second attack to recover the rest of the TPS, or we may simply proceed to an exhaustive search on all the key bits we have not yet found.

An entirely different approach - "memoryless" attacks. Biham and Shamir were not able to turn any of the above approaches into an attack on DES faster than exhaustive search. However, in a later work [34], they were able to come up with a new way to perform the attack that worked with longer TPSes than any of the preceding methods, and to use this in a more powerful attack on DES. It was also intended to use less memory, to be highly parallelisable, and to produce results fast enough to deal with frequent key changes.

The basic idea is that the TPS should be very close in size to the actual key, and that as soon as a given TPS is suggested by one of the pairs, all possible values of the remaining key bits should be tried to see if one of them gives us the correct key. If none of them do,

we resume the attack.

The attack requires a much more complex set of chosen plaintexts than before - instead of a set of pairs with the required input difference, huge "structures" are defined containing several (2^{13} in the attack on DES) pairs with that input difference, all of them related in some particular way. (Biham and Shamir later optimised the attack by using quartets instead of pairs.) In the example given, each structure had an extremely low chance of containing even one right pair, and 2^{35} structures were needed to give the attack a 58% probability of success.

Working out how the pairs in a structure should be related depends heavily on the differential characteristic being used, the cipher being attacked, and any other tweaks made to the method - in the attack on DES, for instance, the characteristic began at the second round and only a fraction of the pairs in each structure had the required difference going into this round. This is something the cryptanalyst will have to work out for herself for each individual attack.

3.3.2 Variants of the differential cryptanalytic attack

There have been several variants of differential cryptanalysis, and attacks building on the basic concept, since it was first introduced. We briefly discuss some of these variants here; *truncated* differential cryptanalysis, *higher-order* differential cryptanalysis, and *multiplicative/hybrid* differentials [42]. Although we do not discuss it here, we also draw the reader's attention to another variant, *impossible* differential cryptanalysis [160, 25, 26].

Attacks building on the concept of differential cryptanalysis include *boomerang attacks* [229], *amplified boomerang attacks* [150] and *rectangle attacks* [28]. Again, however, we do not cover these here.

Truncated differential cryptanalysis

In a conventional differential attack, ΔX and ΔY are completely defined. However, it may be possible to carry out attacks in which we do not need to know the full output difference, just some of the bits in it. For example, given a DES S-box, instead of working with output difference 0110, we might be interested in all output differences in which bit 3 changed and bit 4 did not, and be uninterested in the left-hand bits. So, instead of the differential $(\Delta X, 0110)$, we would instead have the *truncated* differential $(\Delta X, ??10)$

Similarly, we might have discovered for a given Serpent S-box that output difference

0011 occurs with high probability as long as the input difference is either 0111 or 1111 (As far as we know, this is not in fact the case for any of Serpent's S-boxes.) and decide to use both of these input differences. This would give us the truncated differential $(?111, 0011)$.

Both of these are special cases of truncated differential characteristics, in which only some of the bits in the input/output differences are specified. Truncated differential cryptanalysis does, however, generalise even further than this - it is not even necessary to specify specific bits. A truncated differential is defined as a pair (information determining a subset of all input differences ΔX, information determining a subset of all possible output differences ΔY.) For example, the truncated input difference might be ??00, and we might be interested in whether the first and last bits of the output difference XOR to 1.

Truncated differential cryptanalysis was first defined by Lars Knudsen [159], in which it was used to attack 6-round DES. The differential cryptanalysis method we defined for "When the TPS is too long for the first method, and there are too many pairs for the clique method" is in fact a form of truncated differential attack that predates this definition! The "partial differential" cryptanalysis of Biryukov and Kushilevitz [39] is another example of truncated differential cryptanalysis.

Higher-order differential cryptanalysis

Higher-order differential cryptanalysis was first defined by Lai [169], and developed further by Knudsen [159]. To understand higher-order differentials, it is first necessary to understand the concept of *derivatives*:

Definition 3.3.6. The (first-order) derivative of a Boolean function $f(x)$, with respect to a vector s, is defined as $\Delta_s f(x) = f(x+s) - f(x)$. Usually, we will be working over $GF(2)$ and so this will equate to $\Delta_s f(x) = f(x) \oplus f(x \oplus s)$. This generalises directly to the case of multiple-output Boolean functions such as S-boxes.

Definition 3.3.7. The definitions of higher-order derivatives are defined recursively from the above definition - so $\Delta^i_{(a_1,\ldots,a_i)} f(x) = \Delta_{a_i}(\Delta^{i-1}_{(a_1,\ldots,a_{i-1})} f(x))$.

For example, consider the second order derivative $\Delta^2_{a_1,a_2} f(x) = \Delta_{a_2}(\Delta_{a_1} f(x))$:

$$\begin{aligned}
&\Delta_{a_2}(\Delta_{a_1} f(x)) \\
&= \Delta_{a_2}(f(x \oplus a_1) \oplus f(x)) \\
&= f(x \oplus a_1 \oplus a_2) \oplus f(x \oplus a_2) \oplus f(x \oplus a_1) \oplus f(x).
\end{aligned}$$

We then also rely on the following results:

Lemma 3.3.8. *If the cipher operates over $GF(2)$, and if the entries in the vector $(a_1, \ldots, a_i)$ are not linearly independent, $\Delta^i_{(a_1,\ldots,a_i)} f(x) = 0$.*

Lemma 3.3.9. *Let $deg(f)$ denote the algebraic degree of f. Then $deg(\Delta_a f(x)) \leq (deg(f(x)) - 1)$. Note that if f is the zero function, for which the degree is undefined in general but usually defined as $-\infty$, we may have to treat it as a special case - or at least avoid confusion by noting that that $-\infty \leq (-\infty - 1)$.)*

Corollary 3.3.10. *If $\Delta^i_{(a_1,\ldots,a_i)} f(x)$ is not a constant, then f has algebraic degree $> i$.*

(The above corollary is used by Knudsen [159] as the basis of an algorithm that, given a Boolean function on multiple outputs (such as a block cipher) returns a lower bound for its algebraic degree.)

We now address the question of how to use this in cryptanalysis. Because any r-th derivative of a multiple-output Boolean function with algebraic degree r is a constant, a "higher-order differential" with probability 1, using chosen-plaintext structures each of size 2^r, is defined for any round function into which plaintext is input directly. (That is, we assemble the 2^r chosen inputs to the function specified by $\Delta^i_{(a_1,\ldots,a_r)} f(x)$, ensuring that all a_k are linearly independent. The XOR of their outputs is a constant with probability 1, and we need to know beforehand what this constant is). If the decryption of the final rounds will allow us to tell if the sum of the outputs was the predicted constant, then we can carry out partial decryptions for the various TPS values, and eliminate any TPS for which the correct output XOR did not result.

Unfortunately, this does not scale very well as the number of rounds increases. For instance, although Knudsen is able to attack an arbitrary 5-round Feistel cipher in this fashion, there seems no way to extend the attack as described to a Feistel cipher with six or more rounds.

To defeat higher-order differential attacks, cryptographers are advised to avoid using Boolean functions of low algebraic degree as S-boxes or round functions.

Multiplicative and hybrid differentials

Multiplicative differential cryptanalysis [42] works with pairs $(x, x' = ax)$ - so we multiply x by a, instead of xoring it with a bitstring ΔX, to obtain x'.

Before the paper describing multiplicative differential cryptanalysis was published, several ciphers such as IDEA [170], Nimbus [176], xmx [185] and MultiSwap [212] utilised scalar multiplication modulo some value m. There were various reasons for this:

- IDEA's designers believed that mixing different operations over different groups, which were "algebraically incompatible", would provide a high level of security. (The other operations they used were XOR and modular addition.) In the 22 years since the cipher was published, it has taken until this year for an attack on the full cipher [155] to be published; and even so this attack is not one which would be feasible in practice.

- xmx's designers were attempting to produce a fast, compact cipher with as much cross-platform portability as possible. Like the designers of TEA [193] and Salsa20 [21], they believed that avoiding S-boxes and permutations in favour of simple operations that all processors would be able to carry out quickly would be the best way to do this.

- Borisov, Chew, Johnson and Wagner [42] point out that scalar multiplication is hard to attack with traditional differential cryptanalysis using pairs $(x, x + \Delta X)$ or $(x, x \oplus \Delta X)$.

Borisov et al. exploited the fact that m was typically equal to $2^{\text{size of block}}$ or $2^{\text{size of block}} - 1$) to come up with ways to use multiplicative differentials in the cryptanalysis of such schemes. They also looked at generalisations of this - for example the inputs might be values mod m, but the outputs might only be mod q for some $q < m$.

The key to multiplicative differential cryptanalysis is that we are still representing the numbers involved as z-strings of bits, and the modulus used is often either the largest value such a bitstring can represent or that value + 1. This leads to relationships such as the following, which can be exploited by cryptanalysts:

- Where $m = 2^l - 1$, $-x \bmod m = (x \oplus 11 \ldots 1) = (x \oplus n)$.

- Where $m = 2^l$, $(2^k)x \bmod m = (x \ll k)$.

- Again, where $m = 2^l$, reversing the bits transforms $(x, 2x)$ to $(x, x/2)$.

- Where $m = 2^l$ and x is odd, $-x \bmod m = (x \oplus 11 \ldots 10) = (x \oplus (n - 2))$.

Multiplicative differential cryptanalysis also makes use of the following result:

Lemma 3.3.11. *The bitstring representation of any positive integer m can be expressed as a sequence of strings of the form (111 ... 1) or (100 ... 0). For instance, 30777 = 111100000111001 = (111, 100000, 11, 100, 1).*

Interestingly, Borisov et al. also demonstrated a *truncated* multiplicative differential!

Hybrid differentials Where the blocks on which a cipher operates are split into sub-blocks, we may want to use different types of differential on different sub-blocks. In the same paper in which they had published the aforementioned work on multiplicative differentials [42], Borisov et al. presented an example where a 64-bit block *abcd* was split into four 16-bit sub-blocks (a, b, c, d), a multiplicative differential applied to a and d, and a conventional differential to b and c. This resulted in the chosen input pair $(a, b, c, d), (a' = a \times k, b' = b \oplus 5, c' = c \oplus 5, d' = d \times k)$.

These were referred to as "hybrid differentials".

3.4 Stream ciphers based on linear feedback shift registers

The block ciphers with which we have dealt with so far have all acted on large blocks of several bits at a time; DES, for example, encrypting 64-bit plaintext blocks to 64-bit ciphertext blocks. Stream ciphers, by contrast, can encrypt data one bit at a time; the bulk of the stream cipher's operation being to generate a stream of apparently random "keystream" bits which are then xored, one bit at a time, with an arbitrary-length plaintext to generate the ciphertext.

(There do exist stream ciphers which output streams of bytes, or even words, instead of individual bits - but we do not concern ourselves with these as our research is relevant to a particular design outputting a stream of bits.)

The LFSR (Linear Feedback Shift Register) is a component frequently used in the construction of stream ciphers with hardware implementations, such as the eSTREAM finalists SOSEMANUK [17], $DECIM^{v2}$ [18] and Grain [131]. It is not so popular in software-based stream ciphers, for reasons of efficiency which may be due to its operating on individual bits instead of bytes or words.

The LFSR may be represented, as seen in Figure 3.7, as an array of 1-bit integers and a linear function. (There are many ways in which the linear function may be represented,

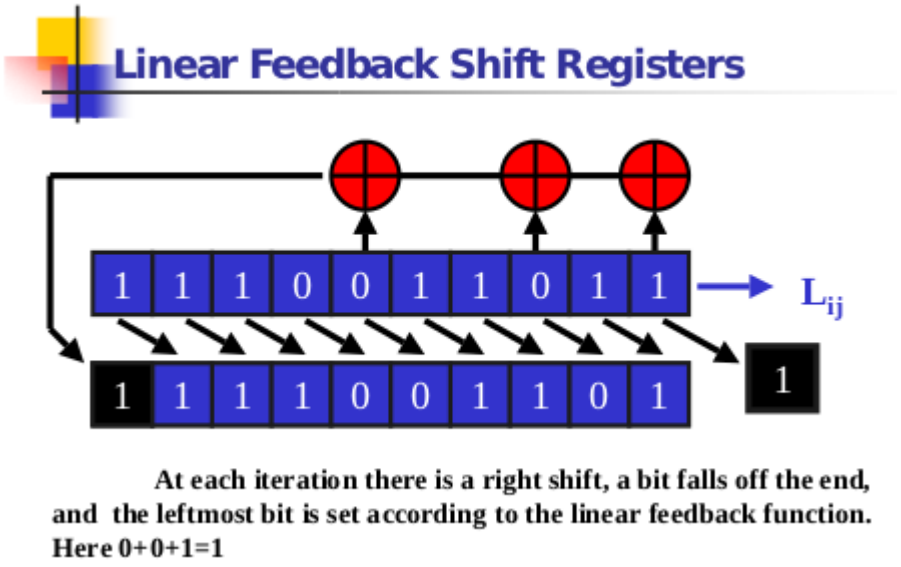

Figure 3.7: Diagram showing the operation of a linear feedback shift register [75].

perhaps as a list of indices of array elements). Let the number of LFSRs in use be denoted n, and let the total length of all the LFSRs in the cipher be denoted k. In a LFSR-based stream cipher, the key is the k-bit integer representation of the initial set of k bits used to initialise the LFSR arrays.

There is a restriction on the key value for this type of cipher; if all of the bits in the initial state of any LFSR are equal to 0, all of that LFSR's output bits will also equal 0, making life much easier for the cryptanalyst! No key such that this is the case is therefore permitted.

The ways in which an LFSR may be used in a stream cipher vary; for simplicity we will consider only the two most basic stream cipher designs to make use of these: *combiner*-based stream ciphers, and *filter*-based stream ciphers. We will first provide a brief description of the workings of an LFSR, then explain how these two models utilise LFSRs.

1. The cryptanalyst initialises the internal states of the LFSRs with the key - the initial values of the individual bits.

2. At each iteration, the values of various bits in the internal states are input to a

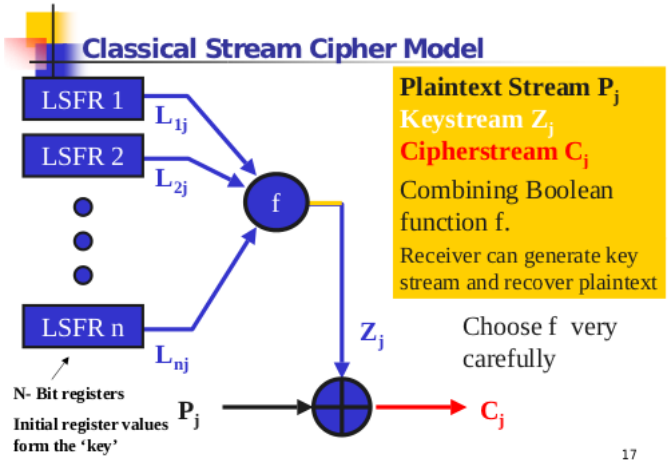

Figure 3.8: Diagram showing the operation of a combiner-based stream cipher [75].

"combiner" or "filter" function, which in turn outputs a bit of keystream.

3. Each LFSR then updates its internal state. A 1-bit value α is calculated by xoring together certain bits within the current state. The entries in the LFSR are then shifted to the right - so for the LFSR in Figure 3.7, array element 9 takes the value previously held by array element 8, element 8 takes the value previously in element 7... and so on.

4. Finally, the value α is placed into $State[0]$.

The particular subset of LFSR elements xored together is extremely important in ensuring that the sequence of LFSR states does not begin to repeat itself too soon. Where there are m elements in the LFSR internal state, a subset corresponding to a so-called "primitive polynomial" ensures that this sequence will have the maximum possible period of $(2^m - 1)$ and will never include the all-zeroes state.

The combiner based stream cipher, at each iteration, inputs the n-bit value consisting of the rightmost bit from each LFSR into a nonlinear *combining function* (or, alternatively, *combiner* function). The LFSRs update their internal states as just described, and the combiner function outputs one bit of keystream. Some ciphers attempt to complicate this procedure with "irregular clocking", in which at least one LFSR will not update its internal state in every iteration, or in which some LFSRs will update more than once in between each keystream bit output.

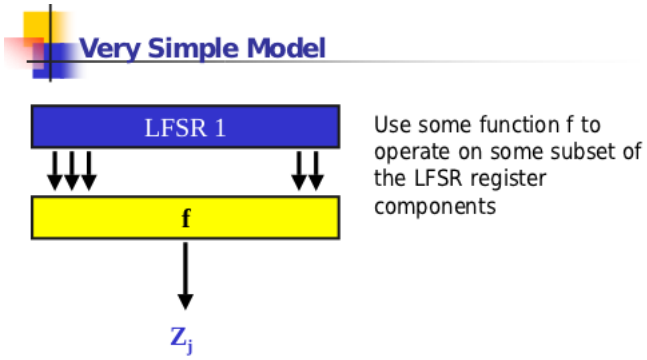

Figure 3.9: Diagram showing the operation of a filter-based stream cipher [75].

A filter based stream cipher uses only one LFSR. It takes several of the state bits, not merely (perhaps not even including!) the rightmost bit as input. As with the combiner model, at each iteration the bits are input to the filter function, which in turn outputs one keystream bit, and the LFSR updates its internal state.

The combiner and filter functions need to satisfy a great many cryptographic criteria, to protect against many different forms of attack. A list of these criteria, and a brief discussion, will be presented in the later chapter in which we attempt to evolve filter functions satisfying said criteria.

In recent years, stream ciphers have tended to move away from the LFSR design. Algebraic attacks [109, 103, 132] have forced significant increases in the number of LFSR state bits that must be input to the filter function, which may be a factor in this, making it more difficult to create adequate Boolean functions for this purpose. In Chapter 4, we discuss reasons why the combiner model had fallen out of favour even before this. In any case, during the eSTREAM contest [35] to design new stream ciphers, many ciphers either did not use LFSRs, or used a mixture of LFSRs and "NLFSRs" (NonLinear Feedback Shift Registers).

In Figure 3.7, at each iteration the new leftmost bit in the LFSR is obtained by xoring certain of the other state bits - a linear function on these bits. In a NLFSR, the new leftmost bit is a nonlinear function of these bits - however, as stated by Schneier [211], the mathematical theory behind such designs is not so well known as for LFSRs, and it is not easy to rule out the possibility of hard-to-detect weaknesses in the keystream generation eventually resulting from such functions.

3.5 A brief introduction to metaheuristics

Metaheuristic search techniques (also referred to as "combinatorial optimisation", and "nature-inspired search techniques") have been applied to a very wide range of problems in several different fields, such as network routing [59], scheduling machine workloads in aluminium foundries [128], creating university timetables [47], and circuit design [157]. They have also been applied to different areas of cryptology - mostly cryptography, in that they have been used to construct cipher components, but also cryptanalysis. We will discuss this in more detail later on, but first of all we will explain the concept of metaheuristics.

In a metaheuristic search algorithm, the user is attempting to solve a problem by creating - or "evolving" - an entity of some form - whether this be a bitstring with particular properties, or a design for an engine component... The user first defines a "cost function" or "fitness function", which takes an entity of the type being evolved, and outputs a scalar value. In algorithms which use cost functions, high quality problem solutions should correspond to low function output values, and poor solutions to high cost values. For algorithms using fitness functions, the opposite is true.

There is no rigid definition of a "metaheuristic" search method. The various search techniques to come under this banner have various features in common, but continuing research has in some cases generalised the definition further.

- Most metaheuristic algorithms, as stated, require the user to rigorously define the class of object being evolved, and then define a cost function C which returns a single scalar value $C(x)$ such that the lower the value of $C(x)$, the better the solution candidate x is considered to be.

  (As stated, for some algorithms, a fitness function F must be defined such that high values of $F(x)$ correspond to high-quality solutions.)

  In the event that there are various criteria which we need to optimise, some of which may conflict with others, *multiobjective evolutionary algorithms* [154] are a generalised form of metaheuristic, in which separate cost or fitness functions are defined for each criterion, and one of various strategies is used to optimise these simultaneously by some more sophisticated method than a single, conventional cost function computing an overall cost from the outputs of the separate functions.

- The algorithm is not deterministic. It uses a pseudorandom number generator

(PRNG) frequently in making decisions, or constructing candidate solutions, and two runs of the same algorithm with different PRNG seed values should both behave differently.

- The algorithm is an iterative algorithm. In each iteration, it constructs either a single solution candidate, or a "population" of several candidates.

- In all iterations other than its first, the algorithm uses information (possibly including the cost) from the candidate solutions built up in some or all of the previous iterations to build a replacement solution or set thereof. While the average cost at iteration i is not guaranteed to be lower than the average cost at iteration $i-1$, over time the cost should decrease.

  There are various approaches to this. Memetic algorithms use the existing population to build new solutions forming a new population, and then make several incremental changes to each population member. In fact, most metaheuristics operate on entities such that small changes with limited effect on the cost can be made to the entity. Simulated annealing uses these changes to continuously change a single candidate solution. Ant algorithms retain information from previous solutions, which they use in constructing the new solutions, but do not retain either the previous solutions or enough information to reconstruct them. Non-generational evolutionary algorithms [11] are similar to memetic algorithms, but only replace a few of the population members with new candidates at each stage.

- It must be possible to define a "search space". This is the set of solution candidates, consisting entirely of entities of the form being evolved. Any function used during the search that generates a new solution candidate, whether from scratch or using information from previous iterations, must have this set or some subset thereof as its codomain. Any function which modifies a candidate solution during the search must also have this set, or some subset thereof (perhaps partially defined by the pre-modification candidate) as its codomain - so the search space is closed under all defined functions which modify candidate solutions. There should be no element within the search space such that the search algorithm cannot at some stage consider it as a candidate.

  In some cases, it may seem (wrongly!) as though this is violated, inasmuch as not every element in the search space actually does correspond to a valid solution to

the problem that the user wishes to solve. It may be that the search space consists of a set of items such that some can be transformed into valid solutions, and the cost function is based on the success of the transformation in achieving this [84]. In some cases, the concept of *genotype-phenotype separation* results in a search space consisting entirely of entities of one type, and the cost function penalises those which cannot be transformed into entities of the second type - as well as rewarding the relevant qualities of the entities of the second type when the transformation succeeds.

- We refer above to "Any function which modifies a candidate solution during the search". Such functions are known as "move functions", and a single application of a move function is a "move". They affect the definition of the search space, in that:
  - Any entity may be viewed as having, for various $x \geq 1$, an "x-move neighbourhood", the subset of the search space consisting of solution candidates which can be obtained from the original entity by making x moves.
  - Where solution S_a is a member of the x-move neighbourhood of solution S_b, S_b must also be a member of the x-move neighbourhood of S_a.

  The definition of a move depends on the entity being evolved. For example, if we were attempting to evolve a bitstring, we could choose one of its bits and flip it. Or we could swap the positions of a zero and a one within the bitstring. A single move should in general only have a small effect on the value of the cost function.

- The algorithm's performance is initially little better than an algorithm choosing elements from the set of candidate solutions at random. While these early solutions may, depending on the algorithm, be subject to some form of (possibly deterministic) optimisation, the early stages of the algorithm are intended to explore a wide region of the search space. Over time, the algorithm's behaviour moves away from exploration, however, and becomes more and more focused on increasing the quality of the solutions within their current region of the search space.

We present two important definitions

Definition 3.5.1. A *local optimum* is a candidate solution such that its cost is less than the cost of any other solution candidate in its 1-move neighbourhood. Precautions must

be taken to prevent the search algorithm becoming "stuck" too early in a local optimum when there may be better local optima, or even the "global" optimum as defined below, to find.

Definition 3.5.2. A *global optimum* is a candidate solution such that its cost, c, is less than or equal to the cost of all other members of the search space. There may be more than one such, depending on the cost function and search space.

3.5.1 Hill-climbing

Whether *hill-climbing* counts as a metaheuristic is debatable. It is used as a component in various metaheuristic search algorithms, and is vital to memetic algorithms in particular, but can also be used as a search algorithm in its own right.

There are many different hill-climbing algorithms, but in general they implement an algorithm which tests some or all of the 1-move neighbourhood of the current solution candidate S, either replacing S immediately whenever a candidate with lower cost is found, or completing the tests and then replacing S with the best improvement found. This is repeated either for some fixed number of iterations, or until no improving candidate is found in one of the iterations.

Hill-climbing is a *local-search*-based metaheuristic, in that the candidate solutions it checks at each iteration are all within the 1-move neighbourhood of the current candidate.

The below pseudocode shows the two hill-climbing algorithms we incorporated into our search algorithms in this paper.

So-called "shotgun" hill-climbing, executing several hill-climbing algorithms from several different, randomly-chosen starting points S_0, and accepting the best result, has been used frequently in cryptanalysing old pencil-and-paper ciphers [129] and machine ciphers - in particular, allowing the cryptanalysis of the German Enigma when a ciphertext is known but none of its corresponding plaintext [222].

Algorithm 1 Pseudocode for a deterministic hill-climbing algorithm

```
S_0 denotes initial candidate
S ← S_0
repeat
    S_best ← S
    ACCEPTS_IN_THIS_LOOP ← false
    for x ← 0, sizeof(1-move neighbourhood of S) do
        S_x denotes the xth member of the 1-move neighbourhood of S.
        cost_diff ← C(S_x) − C(S_best)
        if cost_diff < 0 then
            ACCEPTS_IN_THIS_LOOP ← true
            S_best ← S_x
        end if
    end for
    if ACCEPTS_IN_THIS_LOOP = true then
        S ← S_best
    end if
until ACCEPTS_IN_THIS_LOOP = false
return S
```

Algorithm 2 Pseudocode for a non-deterministic hill-climbing algorithm

```
            ▷ This algorithm was only used in situations where the other was infeasible.
S_0 denotes initial candidate
S ← S_0
repeat
    S_best ← S
    ACCEPTS_IN_THIS_LOOP ← false
    for x ← 0, CANDIDATES_PER_LOOP do
        S_x ← some randomly chosen member of the 1-move neighbourhood of S.
        cost_diff ← C(S_x) − C(S)
        if cost_diff < 0 then
            ACCEPTS_IN_THIS_LOOP ← true
            S_best ← S_x
        end if
    end for
    if ACCEPTS_IN_THIS_LOOP = true then
        S ← S_best
    end if
until ACCEPTS_IN_THIS_LOOP = false
return S
```

3.5.2 Simulated annealing

Simulated annealing is another local-search based algorithm, akin in many ways to a more complex form of hill-climbing. It is inspired by a technique used in metallurgy to eliminate defects in the crystalline structures in samples of metal.

In simulated annealing, some initial candidate solution, S_0, usually chosen at random, is input to the SA algorithm, along with the following parameters:

- The cost function C.
- The initial value T_0 for the "temperature". The higher the temperature in the current iteration, the more likely the search algorithm is to accept a move which results in a candidate solution with higher cost than the current candidate (that is, to store said candidate solution as the "current candidate"). The temperature drops over time, causing the algorithm to accept fewer non-improving moves and hence to shift away from exploration and towards optimisation. Towards the end of the search, it is extremely rare for the algorithm to accept a non-improving move, and its behaviour is very close to that of a hill-climbing algorithm.
- In choosing the value of T_0, various sources state that it should be chosen so that a particular proportion of moves are accepted at temperature T_0. There is very little information or advice available as to what this proportion should be. In one of the earliest papers on simulated annealing [156] it is stated that any temperature leading to an initial acceptance rate of 80% or more will do; however our initial experiments indicated that this was far too high for most of the experiments in this thesis. We usually settled on an initial acceptance rate of 0.5 or 0.6 instead of 0.8.

  Having chosen the initial acceptance rate, the experimenter executes the annealing algorithm with various T_0 until a temperature is found that achieves a fraction close enough to this. We started with the temperature at 0.1, and repeatedly ran the algorithm, doubled the temperature, and re-ran the algorithm until an acceptance rate at least as high as that specified was obtained. Where T_a was the temperature at which this had been achieved, and $T_b = T_a/2$, we then used a binary-search-like algorithm to obtain a temperature between T_a and T_b that would result in an acceptance rate $\approx 50\%$.

- A value α; the "cooling factor", determining how far the temperature decreases at each iteration of the algorithm.
- An integer value: MAX_INNER_LOOPS, determining the number of moves that the local search algorithm can make at each temperature.
- The stopping criterion must also be specified. We used a MAX_OUTER_LOOPS value, indicating how many times the algorithm was to be allowed to reduce the temperature and continue searching before it stopped.
- We also specified a $MAX\_FROZEN\_OUTER\_LOOPS$ parameter. If the algorithm had, at any stage, executed this many outer loops without accepting a single move, it would be considered extremely unlikely to do anything other than remain completely stationary from then on, and would be instructed to terminate early.

3.5.3 Memetic algorithms

Memetic algorithms [183] combine local optimisation with the existing metaheuristic of "genetic algorithms", and have proven to be extremely effective search techniques.

There is some variation in their working - in particular, not every implementation for every problem domain will go through the four main stages in the same order that we do, and some will use more sophisticated machine learning techniques in the local optimisation stage instead of the straightforward hill-climbing we do. With this noted, we continue with our description.

The memetic algorithm maintains a "population" of candidate solutions, in the form of a multiset with size determined by the parameter *popsize*. Over several iterations; or "generations" - analogous to the outer loops of simulated annealing - new populations P_i will be derived from their immediate predecessors. Members of the population P_0 at the start of the algorithm are generated randomly and hill-climbed to local optima. In our implementation, an "interim" multiset contains the results of applying the various stages of the algorithm to P_i - for the purposes of this chapter we denote this set PCP (Post-Crossover Population). PCP is cleared at the start of each generation, and repopulated by the "crossover" operation in said generation. The members of PCP are then randomly altered during the "mutation" phase of the iteration, hill-climbed, and, during the "selection" phase, used to generate P_{i+1}.

Algorithm 3 Pseudocode for simulated annealing algorithm

```
S ← S_0
bestsol ← S_0
T ← T_0
ZERO_ACCEPT_LOOPS ← 0
for x ← 0, MAX_OUTER_LOOPS − 1 do
    ACCEPTS_IN_THIS_LOOP ← false
    for y ← 0, MAX_INNER_LOOPS − 1 do
        Choose some S_n in the 1-move neighbourhood of S.
        cost_diff ← C(S_n) − C(S)
        if cost_diff < 0 then
            S ← S_n
            ACCEPTS_IN_THIS_LOOP ← true
            if C(S_n) < C(bestsol) then
                bestsol ← S_n
            end if
        else
            u ← Rnd(0, 1)
            if u < exp(−cost_diff/T) then
                S ← S_n
                ACCEPTS_IN_THIS_LOOP ← true
            end if
        end if
    end for
    if ACCEPTS_IN_THIS_LOOP = false then
        ZERO_ACCEPT_LOOPS ← ZERO_ACCEPT_LOOPS + 1
        if ZERO_ACCEPT_LOOPS = MAX_FROZEN_OUTER_LOOPS then
                                          ▷ Algorithm terminates early.
            return bestsol
        end if
    end if
    T ← T × α
end for
return bestsol
```

The number of generations is one of the parameters - $NO\_OF\_GENERATIONS$, analogous to the MAX_OUTER_LOOPS parameter of simulated annealing.

The crossover function, $cross(p_1, p_2)$, takes two "parent" candidate solutions p_1 and p_2 as input, and outputs a "child" candidate solution o_1 which in some way combines features from both p_1 and p_2. Note that $o_1 = cross(p_1, p_2)$ is not necessarily equal to $o_2 = cross(p_2, p_1)$.

Several different crossover algorithms have been designed for the evolution of bijective functions (or, indeed, any entity representable as a permutation on a set of integers), and it is considered extremely important to choose a good crossover algorithm for the problem domain. In the section of this thesis in which we attempt to evolve bijective functions over the finite field $GF(2^n)$, we compare two different crossover methods; PMX ("Partially Mapped CROSSover") and cycle crossover [178]. These two crossover methods were chosen because of their focus on the values of x mapping to each output, instead of the order in which these outputs appeared.

- *Cycle crossover* works as follows

  PARENT 1:a b c **d** e f g h i j

  PARENT 2:c f a **j** h d i g b e

  (Randomly chosen cycle start point is marked in bold.)

  The element of Parent 1 at the cycle start point is copied into the child in the same position:

  CHILD:? ? ? d ? ? ? ? ? ?

  The element in the same position in Parent 2 is the next to be copied into the child. However, it is copied in into the same position in which it occurs in Parent 1:

  CHILD:? ? ? d ? ? ? ? ? j

  This process continues until the process returns us to the original cycle start point - in other words, when a "loop" or "cycle" has been created. In this case:

$(d, j) \rightarrow (j, e) \rightarrow (e, h) \rightarrow (h, g) \rightarrow (g, i) \rightarrow (i, b) \rightarrow (b, f) \rightarrow (f, d) \rightarrow (d, j)$ again.

```
CHILD:? b ? d e f g h i j
```

Any still-vacant positions in the child are then filled by copying in the corresponding values from Parent 2:

```
CHILD:c b a d e f g h i j
```

- *PMX crossover* begins by choosing two "crossing points" at random, as illustrated by the vertical lines in the below. The elements of Parent 1 between these points are copied into the child:

```
PARENT 1:a b | c d e f | g h i j
PARENT 2:c f | a j h d | i g b e
   CHILD:? ? | c d e f | ? ? ? ?
```

Next, any elements of Parent 2 which have not already been copied into the child are copied in:

```
PARENT 1:a b | c d e f | g h i j
PARENT 2:c f | a j h d | i g b e
   CHILD:? ? | c d e f | i g b ?
```

C was already copied in from Parent 1, so the element in position [0] cannot be equal to C. We see that the Parent 2 element in the same position as C in Parent 1 is A, and copy that into position [0]. Similarly, the final element cannot equal E, and so we put H in that position, the Parent 2 element in the same position as the E of Parent 1.

```
PARENT 1:a b | c d e f | g h i j
PARENT 2:c f | a j h d | i g b e
   CHILD:a ? | c d e f | i g b h
```

The final unallocated position is trickier. We cannot copy F in, as it is already present in the child. We look for F in Parent 1, and find D in the corresponding position of Parent 2. Unfortunately, D has also been copied into the child by this point! We go on to look for D in Parent 1, and find that J is present in the same position of Parent 2 and has not been copied into the child, allowing us to complete the process:

```
PARENT 1:a b | c d e f | g h i j
PARENT 2:c f | a j h d | i g b e
   CHILD:a j | c d e f | i g b h
```

Whichever crossover method we choose, the following two parameters are involved:

- *no_of_children*: When p_1 and p_2 are selected from P_i, this determines whether, if the crossover function is applied, it will merely be used to add $o_1 = cross(p_1, p_2)$ to PCP, or whether $o_2 = cross(p_2, p_1)$ will also be calculated and added.

  If the crossover function is not applied, this determines whether p_1 alone, or both p_1 and p_2, are added to PCP.

- *crossover_probability*: When p_1 and p_2 are selected from P_i during the crossover phase, this determines the probability of the crossover function being applied - i.e whether o_1 (and o_2, depending on the previous parameter), or p_1 and perhaps p_2, are added to PCP in this generation.

We also need a "mutation function", $mutate(c)$, taking a candidate solution from PCP as input, making a small random change (the "mutation") of some form to it, and returning the result (which replaces the original in PCP). In our experiments, the

mutation function makes one move as defined by the same local search methodology used for simulated annealing and hill-climbs; in the case of evolution of bijective functions, this means that two truth table elements are swapped. Two parameters are relevant to this:

- *max_possible_mutations*: During the "mutation phase" of the algorithm, this defines the maximum number of mutations that may be applied to any single candidate.
- *mutation_probability*: Each potential mutation (up to the number defined by the criterion above) occurs randomly with this probability, independent of the other potential mutations.

Mutation adds an aspect of exploration into the memetic search, enabling it to escape from local optima.

The third phase, hill-climbing, is almost identical to the hill-climbing algorithm as defined before. Note, however, that since memetic algorithms utilise a fitness function instead of a cost function, the algorithm must be tweaked to accommodate this. In this phase, the members of PCP are all hill-climbed to local optima with respect to said fitness function.

Finally, we have the "selection" phase, which is itself divided into various subphases. The implementer may decide to sort the elements of PCP by their fitness values for the sake of efficiency at the start of the selection phase, if so this sorting is the first subphase.

After the sorting is carried out (or not), the next subphase is the "elitism" subphase. If the parameter *elitism_level* has a nonzero value, the *elitism_level* members of P_i with the highest fitness values are copied directly into P_{i+1}. If this results in a full population (which is not advisable!) the selection phase ends. If not, we need to use a selection method to keep choosing elements from PCP to add to P_{i+1}.

Let $|PCP|$ be denoted M. The two selection methods we experiment with in this thesis are:

1. **Roulette-wheel selection:** This method requires the fitness function to output a value ≥ 0 for all possible inputs. Let $\sum_{i=0}^{M} fitness(c_i)$ be denoted Z. Let the number of places remaining in the population be denoted r. Then we follow the procedure in the pseudocode for Algorithm 4:

Algorithm 4 Pseudocode for roulette-wheel selection

for $i \leftarrow 1, r$ **do**
  One member of PCP is selected at random with probability $fitness(c_i)/Z$
    ▷ All r selections are independent and at random.
      ▷ A candidate may be selected more than once.
  A copy of this member is added to P_{i+1}.
  The original member is placed back in PCP.
end for

2. **Rank selection:** For this selection method, the members of PCP must be sorted by fitness. Indexing from 1 upward, $PCP[1]$ is the candidate with the lowest fitness; $PCP[M]$ the candidate with the highest.

   As before, in each of r independent trials, a candidate is selected from PCP. A copy of this candidate is placed in P_{i+1}, and the candidate is replaced in PCP. The difference between this and roulette wheel selection is the probability with which each candidate is chosen:

$$P(PCP[i]) = \frac{2i}{M(M+1)}$$

3.5.4 Ant algorithms

The first ant colony optimization method was Ant System, originally described [120] as a metaheuristic that might be applied to the Travelling Salesman Problem (TSP). Later refinements produced the more effective Ant Colony System [119] [175], which took a more elitist approach and achieved superior results against the TSP.

Any problem to which ant algorithms can be applied must be possible to represent as a graph. For the S-box experiments, the graph nodes are the values of x, and the graph is directional - an edge from node x to node y signifies that $S(x) = y$. Furthermore, each edge carries with it a cost - and unlike the conventional TSP, the edge leading from y to x may not have the same cost as that from x to y (making our problem more akin to the Asymmetric TSP).

The problem should also allow a useful cost function to be devised such that, during the construction of each candidate solution, the cost starts at zero and is increased whenever a new component is added until the final cost is derived. The component should be representable as an edge on the graph.

Algorithm 5 Pseudocode for memetic algorithm

```
                                                    ▷ Stage 1: Crossover.
Reset PCP to an empty multiset.
while size(PCP) < POST_CROSSOVER_POPULATION_SIZE do
    Choose p1 and p2 from Pi uniformly at random.
    if Rnd(0,1) < crossover_probability then
        o1 ← cross(p1, p2)
        Add o1 to PCP
        if no_of_children = 2 then
            o2 ← cross(p2, p1)
            Add o2 to PCP
        end if
    else
        Add p1 to PCP
        if no_of_children = 2 then
            Add p2 to PCP
        end if
    end if
end while
                                                    ▷ Stage 2: Mutation
for i ← 0, POST_CROSSOVER_POPULATION_SIZE − 1 do
    for j ← 0, max_possible_mutations − 1 do
        if Rnd(0,1) < mutation_probability then
            Apply one move (as defined for local search) to PCP[i]
        end if
    end for
end for
                                                    ▷ Stage 3: Hill-climbing
for i ← 0, POST_CROSSOVER_POPULATION_SIZE − 1 do
    Hill-climb PCP[i] to a local optimum.
end for
                                                    ▷ Stage 4: Selection.
Reset the population to the empty multiset.
if elitism_level is specified then
    copy elitism_level members of Pi into Pi+1
end if
while size(population) < popsize do
    use a selection function to choose the next member of PCP to add to Pi+1.
end while
```

The value d_{ij} denotes the amount by which the cost is increased if the edge from node i to node j is added, i.e. if $S(i)$ is assigned the value j. While for some problems (such as the TSP) d_{ij} is constant, here it is affected by the truth table values that have already been assigned, and so must be recalculated every time we need to add a value for $S(i)$.

The following parameters are involved:

- The particular type of ant algorithm. In our experiments, we compared Ant System, Dorigo's original Ant Colony System, and the version of ACS defined in Sean Luke's "Essentials of Metaheuristics" [175]. Other algorithms exist; for instance "AntNet" [59, 60], a specialised variant designed for network routing problems.

- *hillclimb_trails*. This is a boolean value which determines whether or not local optimisation (i.e. hill-climbing the constructed solutions) is used during trail-building.

- *next_index_method*. After adding an edge from i to j, this parameter determines which node the ant should try to add an edge leading from next. In this thesis, we experiment with "cycle", in which the next node is node j, and "increment", in which the next node is node $(i + 1)$.

- α and β are floating-point values. The value of α determines the amount of influence pheromone levels have on edge selection, and the value of β determines the influence of d_{ij}.

- e - the elitist pheromone update parameter (Used only in ACS versions of the global update stage.)

- τ_0 - the initial amount of pheromone on each edge.

- no_of_ants - the number of ants.

- Q - a scalar value by which the amount of pheromone deposited in the global update is multiplied. In the paper in which Ant System was originally described [120], after experiments with $Q = 1$, 100, and 10000, 100 was accepted as the "experimentally determined optimal value". However, in later descriptions of ACS [119, 121, 175], $Q = 1$ was implicitly used, and no other values were mentioned.

- q_0 - for ACS algorithms, this dictates the probability that a given edge selection will use an elitist selection method instead of the exploratory Ant System method. For Ant System, q_0 is in effect always zero.

- ρ - the non-elitist pheromone update parameter (also known as the "evaporation rate").

In the below pseudocode, which describes all three of Ant System, Dorigo's ACS, and Luke's ACS, τ_{ij} denotes the amount of pheromone on the edge corresponding to $S(i) = j$. An ant trail is deemed to be complete when every node has an edge leading from it; i.e. when every i has been assigned an output value $j = S(i)$.

3.5.5 Metaheuristics in cryptography

Metaheuristics have been used to evolve Boolean functions on up to 10 variables with better cryptographic properties than hitherto obtained [89] [80] [84] [82] [88], sometimes using innovative techniques such as searching for the cost function [87]. They have also been used to look into the feasibility of detecting, or guaranteeing the absence of, backdoors in cryptographic Boolean functions [85].

As well as cryptographic primitives, they have also been used to evolve cryptographic protocols [83, 81, 64]. For an excellent survey of the field up to 2001, the reader is urged to read John Clark's DPhil thesis [74]. Clark has coauthored the majority of the papers mentioned so far, and is probably the most significant figure in research into the use of metaheuristics in cryptology.

There has also been research into the evolution of quantum algorithms, circuits, and protocols (collectively referred to as "quantum computing artefacts" [92]). Given the relevance of quantum computers to cryptanalysis, in particular the effects of Shor's and Grover's algorithms on cryptosystems currently believed secure (should quantum computers ever be implemented) and the research field of *post-quantum* cryptography that has evolved in response to this, we do not think it beyond the remit of this section to briefly discuss these.

The most recent survey paper in this area [96] was written by Clark and Stepney in 2007, and is an updated version of an earlier survey [95] by the same authors. Many of the most-cited works in the field have been coauthored by Lee Spector [12, 14, 15, 13, 219], frequently in collaboration with Howard Barnum and Herbert Bernstein. A large body of work on the topic has also been conducted at York [90] [91] [44] [177].

As well as cryptology, we also note that there appears to be a large body of literature on the use of metaheuristics, in particular artificial immune systems, in another aspect of computer security; intrusion detection. We have not investigated these papers as they are

Algorithm 6 Pseudocode for ant algorithms

Set the amount of pheromone on each graph edge to τ_0.
best_solution $\leftarrow$ some randomly generated solution candidate.
for $x \leftarrow 0, no\_of\_iterations - 1$ **do**
▷ Each ant builds a trail
Clear all ant trails
(remove all edges, set current nodes of all ants to 0.)
while ant trails incomplete **do**
for $a \leftarrow 0, no\_of\_ants - 1$ **do**
Let i_a denote ant a's current node.
Let the set of unassigned output values at this point be denoted U.
▷ Add an edge from i_a to some node $j_a \in U$
▷ (based on the cost of the edge and level of
▷ pheromone on it).
$q \leftarrow rnd(0, 1)$
if $q \leq q_0$ **then**
Choose node j_a where j_a is the value of k corresponding
to $max_{k \in U}(\tau^{\alpha}_{i_a k} \cdot d^{\beta}_{i_a k})$
else
Node j_a is chosen from the set U with probability:
$\frac{(\tau^{\alpha}_{i_a j_a} \cdot d^{\beta}_{i_a j_a})}{\sum_{k \in U}(\tau^{\alpha}_{i_a k} \cdot d^{\beta}_{i_a k})}$
end if
if ant method is Dorigo's original ACS **then**
▷ Decrease pheromone levels on chosen edge (local update):
$\tau_{i_a j_a} \leftarrow (1 - \rho) \cdot \tau_{i_a j_a} + \rho \cdot \tau_0$
end if
▷ Update current node:
if $next\_index\_method = CYCLE$ **then**
$i_a \leftarrow j_a$
else if $next\_index\_method = ITERATE$ **then**
$i_a \leftarrow (i_a + 1)$ modulo no_of_nodes
end if
end for
if *hillclimb_trails* **then**
Hill-climb all constructed solutions represented by the ant trails
to local optima.
end if
end while
Let *best_iteration* be the ant which constructed the best solution in this iteration.
Let *best_itera_sol* be that solution.
if $cost(best\_itera\_sol) < cost(best\_solution)$ **then**
best_solution $\leftarrow$ *best_itera_sol*
end if
Update pheromone levels on all edges (global update).
(The method varies depending on the choice of ant algorithm).
end for
return *best_solution*

beyond the remit of this document; nevertheless we refer the interested reader to Clark's list of relevant articles [73].

3.6 Cryptanalysis with metaheuristic search.

In the previous section, we presented a brief survey of the application of metaheuristics to cryptography. Despite its substantial contribution in this field, however, metaheuristics has made very few contributions to the other component of cryptology - cryptanalysis. It has been successfully used in the cryptanalysis of some early (already broken) pencil-and-paper ciphers [94] [93], and in some (already obsolete) mechanical cipher machines from the early 20th century. Modern-day ciphers from the computer age, though, have barely been affected by it. The only notable success metaheuristics has achieved in this field is in the cryptanalysis of a proposed identification scheme based on the *Permuted Perceptron Problem* (described below); a problem which had its roots in artificial intelligence and which metaheuristics therefore represented an obvious attack vector for.

3.6.1 Metaheuristic attacks on knapsack ciphers

There have been various attempts to attack knapsack cryptosystems using metaheuristics. However, these [220, 165, 210] have only ever succeeded in breaking problem instances with unrealistically small parameter sizes, whereas more conventional cryptanalytic techniques have been deployed to devastating effect against most of the knapsack variants. We do not consider this to be a promising research direction; partly due to the lack of existing success and partly because it is not likely that knapsack ciphers will see use in the near future. For a slightly more detailed summary of this topic, the reader is referred to Clark's thesis [74]. The criticisms and related information in articles by Rubin [209], and by Bergen et al. [72] may also be of interest.

3.6.2 The Permuted Perceptron Problem

Possibly the most interesting application of metaheuristics to cryptanalysis has been in the attacks on a set of zero-knowledge identification schemes proposed by Pointcheval [205]. These were based on the difficulty of the "Permuted Perceptron Problem"; a harder version of the NP-complete "Perceptron Problem":

Definition 3.6.1. Let A be a non-invertible $m \times n$ matrix (It was recommended by Pointcheval that $n \approx m+16$) with all its entries $\in \{-1, 1\}$. The Perceptron Problem (PP) is the problem of finding a column vector V with all its entries $\in \{-1, 1\}$ such that every entry of $AV \geq 0$.

(In two other papers by Pointcheval [204] [206], the Perceptron Problem is defined merely as the simpler problem of discovering whether such a V exists. In the latter paper, it is shown that even this simpler problem is NP-complete!)

Definition 3.6.2. Let the matrix A be defined as before. Let M be some multiset of integers ≥ 0. The Permuted Perceptron Problem (PPP) is the problem of finding some column vector $V \in \{-1, 1\}^n$ such that the multiset of entries in AV is the multiset M.

Since every solution to the PPP is also a solution to the PP, we see that the Permuted Perceptron Problem is harder than the Perceptron Problem.

Pointcheval's schemes are based on the difficulty of a weaker version of the Permuted Perceptron Problem; whereby it is guaranteed that at least one solution V exists.

Pointcheval provides the following method to generate instances of the PPP for use in these schemes:

1. Let p denote the number of elements of the finite field in which we operate. Decide on the values for m and n. As stated, we probably stick to the restriction that $n \approx m + 16$.

   In addition, there is the question of whether n and m should be odd. It is not explicitly stated that they should, however all values for these in Pointcheval's paper are odd, and an attempt to deal with finite field arithmetic in §4.2 appears to assume that n is odd. However, this in turn would imply that entries of AV cannot be zero; and Pointcheval does not rule 0-entries out. Pointcheval and Poupard do however rule out even values of m and n [206], citing unspecified "technical reasons".

   Pointcheval suggests (m, n)-values of (101, 117), (121, 137), and (151, 167).

2. Generate a random vector $V \in \{-1, 1\}^n$.

3. Generate a random $m \times n$ matrix A with all its entries $\in \{-1, 1\}$.

4. Compute AV. If the ith entry of AV is negative, multiply every entry in the ith row of A by -1 and then recalculate AV with the new A.

We will describe the first of the schemes [205] here. It is described as a "three-pass" protocol. In an x-pass protocol, the first pass is deemed to begin when the first transmission of information is made, and the $(i+1)$st pass begins when a participant P sends a transmission that depends in some way on information P had received during the ith pass. (This transmission is considered to take place during the $(i+1)$st pass.)

(Usually we refer to "rounds" instead of passes; however Pointcheval uses "rounds" to refer to repetitions of the protocol - so by this definition the number of rounds is the number of times the protocol must be repeated until P(the prover can succeed this many times without knowing the secret key) is sufficiently low.)

In this protocol, the participants are Peggy (the prover) and Victor (the verifier). Peggy's private key consists of the vector V as described above. Her public key consists of the matrix A and the multiset M, and she wishes to prove to Victor that she knows V without revealing any information about it. Also publicly known (or, at least, known to both parties) are the values n and m, and H, some collision-free cryptographic hash function acceptable to both parties.

The following describes a round of the protocol:

Before transmissions begin, Peggy computes

- P, an $m \times m$ matrix such that PA can be obtained by applying some permutation to the rows of A.
- Q, an $n \times n$ matrix such that AQ can be obtained by applying some permutation to the columns of A, and then multiplying some of them by -1.
- W, a random n-vector.
- $A' = PAQ, V' = Q^{-1}V$, and $R = W + V'$.
- $h_0 = H(P|Q), h_1 = H(W), h_2 = H(R), h_3 = H(A'W), h_4 = H(A'R)$. (Note. h_0 could have been defined as $(H(P), H(Q))$ instead. We are not sure if this would be preferable, but it would be more closely related to what is being committed to.)
- **Pass 1:** Peggy sends $(h_0, h_1, h_2, h_3, h_4)$ to Victor.
- **Pass 2:** Victor chooses some random integer $c \in \{0, 1, 2, 3\}$ and sends c to Peggy.

  While Victor did not depend on any information already transmitted to compute c, he could not send it until Peggy's transmission of the hash values had committed her

to the corresponding data. Otherwise this would have supplied her with information useful in pretending to know V when in fact she did not, since she would know in advance what to transmit in the next pass.

- **Pass 3:** If c is equal to:

  0 : Peggy sends (P, Q, W) to Victor. Victor checks that $h_0 = H(P|Q)$, $h_1 = H(W)$, and $h_3 = H(PAQW)$.

  1 : Peggy sends (P, Q, R) to Victor. Victor checks that $h_0 = H(P|Q)$, $h_2 = H(R)$, and $h_4 = H(PAQR)$.

  2 : Peggy sends $(A'V', A'W)$ to Victor. Victor checks that $h_3 = H(A'W)$, $h_4 = H(A'V' + A'W)$, and that the multiset of the entries of $A'V'$ is equal to M.

  3 : Peggy sends (V', W) to Victor. Victor checks that $h_1 = H(W)$, $h_2 = H(V' + W)$, and that $V' \in \{-1, 1\}^n$.

It would take too long, and be beyond the scope of this section, to explain how with each round of the protocol the chance that Peggy could have passed that many rounds without knowing V decreases. We advise the interested reader to imagine a situation where Peggy knew beforehand that a particular value of c would not occur, and to look at how easy it would then be to construct data which would pass the test for any of the other values. The reader should then look at how hard it is to find a way of further contriving the data, without knowing V, so that it could also pass the test for the "forbidden" value of c.

After a sufficient number of rounds of the protocol, Peggy should have convinced Victor that she knows a solution to the Permuted Perceptron Problem. Pointcheval provides instructions on setting the problem up in a way that will (hopefully) minimise the chance that any other solution exists and if not, will hopefully minimise the number of other PPP solutions that exist.

Attacking the Permuted Perceptron Problem with metaheuristics

Note that A should be non-invertible - it certainly will be if $n = m + 16$, and that with less rows than columns there are not enough simultaneous linear equations to recover V from AV (Even if we knew AV instead of just the multiset of its elements.) Pointcheval looked

at various deterministic methods to solve similar schemes, such as Gaussian elimination, but could not apply them here.

Pointcheval then looked at attacks based on the *majority vector* for the matrix A.

Definition 3.6.3. Given an $m \times n$ matrix with all its entries in $\{-1, 1\}$, the majority vector for that matrix is an n-vector, the ith entry of which is 1 if more than half of the entries in the ith column of the matrix are 1. If this is not the case, the ith entry is -1.

It is not absolutely clear from Pointcheval's work as to whether this resulted from the nature of the PPP or from the way in which problem instances were generated, but nevertheless for problem instances generated as described earlier Pointcheval was able to prove that approximately 80% of the entries of the majority vector were the same as the corresponding entries of V. (It is also the case that the higher the difference between the number of 1s and -1s in the jth column, the more likely the jth entry in the majority vector is to agree with the jth entry of the secret vector.)

Pointcheval considered attacks where approximately 20% of the entries of the vector were flipped and the results tested to see if they were PPP-solutions. (He also considered tweaking this attack so that vector entries were more likely to be flipped if the sum of the values in the corresponding column of A was close to zero.) This fact forced a lower bound of 95 to be imposed on n to ensure a work factor of 2^{64} (we would try to ensure a higher work factor nowadays) in solving the problem; but problem-instances with such a value of n could still be feasibly generated and used.

Pointcheval was aware from the start that metaheuristics might provide a means of solving the problem, partly since the Perceptron Problem was derived from a related problem in artificial intelligence. (In fact, a genetic algorithm [16] which could solve this problem for $m = \mathcal{O}(n)$ was published in the same year as Pointcheval's original paper, though we do not know if he was aware of it.) His approach was to use simulated annealing to search for solutions to the Perceptron Problem. Whenever a solution was found, the algorithm would then test it to see if it was a solution to the PPP. He did not specify the neighbourhood used, but the obvious neighbourhood would be the one in which the neighbours of a given vector were those which differed from it in precisely one entry. Subsequent work [163] [79] did in fact define the neighbourhood thus.

This approach was more successful than the majority vector attacks, but not by enough to break all feasibly-usable problem sizes. The (n, m)-values originally suggested by Pointcheval were based on parameters for which this approach had failed but which

still allowed the identification protocol to be carried out efficiently.

Knudsen and Meier's work Some progress was made in the cryptanalysis of the PPP schemes in 1999 by Knudsen and Meier [163]. They came up with a new cost function that would allow them to attack the PPP directly with simulated annealing, instead of just the PP, and also came up with a modified attack that proved to be more effective against the PPP.

Although they suggested several variations, their method was, basically, the following:

1. Using the new cost function, attack the PPP with simulated annealing t times. Record the result of each attack as the vector which achieved the lowest value of the cost function for that attack. (Most versions of simulated annealing would treat this as the result anyway.)

2. Identify any vector entries which are the same in all t solutions.

3. Begin a new set of t simulated annealing attacks in which the initial state vectors have, for these entries, the values that were found by the previous runs. The values for other entries are chosen at random. As a variant, the attacker may choose to fix these entries at the values found for the remainder of the attack, and to handle errors in these values later on.

4. Return to Step 2. Continue in this fashion until either a solution is found, or the number of entries upon which the searches do not agree is sufficiently small to brute-force. Probabilistic information derived from the previous results can be exploited to speed up the brute force stage, to the extent that the simulated annealing stage may only need to find a quarter of the vector entries!

5. The solution found should be a PP-solution due to the nature of the new cost function. If it is not a PPP-solution, start again.

It should be noted that Knudsen and Meier did not have a way to correct the aforementioned errors, however the ability to start again if the solution was not a PPP-solution seems to have been enough to handle this. The variant could be made to succeed with time complexity $\approx 2^{56}$ against Pointcheval's smallest parameters (They did not in fact expend this much time; this was based on how much faster PP-solutions could now be found.) instead of the original estimate of 2^{64}.

They also noted that in some cases the algorithm would find solutions a lot faster than this. To examine this further, they conducted further experiments in which the algorithm was aborted as soon as it fixed a vector entry at an incorrect value. Using brute force to find the last entries, it was noted that approximately 9% of the time this attack would find the correct solution, and the largest amount of time any of these successful runs were observed to take in their experiments was 2^{52} basic operations. It was also conjectured that these results might further be improved upon by exploiting probabilistic properties of the correct solution (such as correlation with the majority vector) in choosing initial values and checking candidate solutions, and noted that in other instances of the problem the algorithm might have found an alternative PPP-solution (if one existed) had they not aborted it.

Knudsen and Meier concluded by pointing out that their results had used a relatively low starting temperature, and a relatively high cooling rate, during the simulated annealing, and that in a serious cryptographic attack the attacker would probably not use such parameters, having more time available to launch a more effective attack. They recommended that $(m = 101, n = 117)$ should not be used (this recommendation appears to have been widely accepted), and pointed out that for the higher parameters recommended the PPP-based schemes did not offer a significant efficiency advantage over those based on other hard problems.

Clark and Jacob's work Knudsen and Meier had made headway against the PPP by not treating the optimisation-based attack as "black box", but observing what happened during it and using this information to improve the attack. Clark and Jacob [79] drew parallels between this and the way in which side-channel attacks had moved away from viewing cipher-implementations as "black-box" to exploit information leaked by the computations. Their improved attack followed two side-channel-analogous approaches:

- **Fault injection:** Analogous to the concept of causing errors in the cryptosystem's computations and exploiting them, Clark and Jacob introduced "errors" into the cryptanalysis by changing the cost function so that the solution which minimised the cost was, not the correct solution, but one which was closely correlated to it. Several such cost functions were used, along with tweakable parameters allowing the cost to be multiplied by and/or raised to the power of some scalar. Simpler methods such as hill-climbing would then be used to find the correct solution from the related

solutions.

This technique was referred to as "Problem Warping".

- **Timing attack:** Knudsen and Meier had focused on vector entries which all annealing runs agreed on. Clark and Jacob noted that certain entries would, relatively early on in the search, take a particular value and then not subsequently deviate from it. These values tended to be the correct values for those entries. It therefore became important to keep track of how quickly the various entries became "stuck" (later entries might just be getting stuck due to convergence on a local optimum).

  This was inspired by a similar phenomenon observed before when using simulated annealing in non-cryptographic contexts. While not becoming fixed early on, there were situations where, if annealing was being used to find an optimal value for some bitstring, certain bits which had the same value in all optimal solutions (these were known as *persistent variables*) would be such that, as the temperature approached zero, their average value over all the iterations at temperature T_i would converge on their correct value as i increased. A 1995 paper [63] had exploited this by, at each change in temperature, fixing the values of any bits which had taken one value sufficiently more than the other. (The thresholds used to define "sufficiently more" did not change as the temperature decreased.) The idea behind this was that, since the bits were clearly converging on these values, they should be fixed at them to reduce the number of non-improving moves available to the algorithm, and thus increase its efficiency (especially at lower temperatures.) The new approach was known as *thermostatistical persistency* (TP).

  Clark and Jacob, varying this according to their knowledge of the PPP, now applied it to cryptanalysis. They did not attempt to fix the values of these bits until the hill-climbing stage (how long a bit would need to stay at a given value before being considered "stuck" is not discussed - but to use an approach closer to TP, we would need to have a strategy for this). Instead, they looked at observing over multiple runs which bits tended to become stuck soonest and which/how many of these should be considered correct in the search for the overall solution.

Problem warping Part of the inspiration for problem warping had been a property of the multiset M not hitherto used, or indeed mentioned in previous papers. After V and

the initial version of the matrix that would become A (call it A_{init}) had been generated, the values in the multiset of entries of $A_{\text{init}}V$ were binomially distributed with a mean of zero (which, for odd n, would not in fact occur as a value) - and hence the lower the magnitude of a given odd number, the more likely it was to feature in the multiset. The way in which A was derived from A_{init} - by flipping the entries in any row such that $A_{\text{init}}V$ was negative - meant that the lower the magnitude of the positive integer between 0 and n, the more often it could be expected to appear in the multiset, since it occurred as many times as the total number of times it and its negative had occurred before.

This fact was not itself cryptographically useful, in fact it was causing problems for the annealing-based search. Flipping an individual element of the candidate solution caused every entry in $U = AV_{\text{candidate}}$ to change by ± 2, and this was observed to be causing several small positive values in U to become negative when attempts were made to move to a neighbouring candidate - which was in turn causing the search to become too easily stuck in local optima.

Clark and Jacob, apparently counter-intuitively, introduced the parameter $K \in \mathbb{N}$ into the cost function. Instead of simply penalising negative entries in U, any entries below K would now be penalised, causing the search to seek out "solutions" such that $AV_{suggestion}$ would contain only positive values above a certain threshold.

Searches using the modified cost functions corresponding to the various values of K would not find V, as vectors with the high number of 1s and other low-magnitude natural numbers in the correct V would not minimise the cost. However, the solutions resulting from these searches would frequently differ in ≤ 10 places from solutions to the Perceptron Problem, and these solutions could be easily found by a hill-climbing algorithm flipping individual entries. Clark and Jacob's strategy was to attempt the search using ten runs with every possible cost function, and to use brute-force on every result under the assumption that a certain upper bound applied to the number of incorrect bits (this bound being determined by their experiments on various parameter sizes.) For some of the cost functions, this would fail; but for others, it would succeed. And extra information could be gained by noting that any entries such that every problem instance had agreed on their values (or that a heavy majority had) were likely to be correct. The less agreement there was on a particular entry, the more likely it was to be flipped during hill-climbing.

K was incorporated into Knudsen and Meier's cost function to allow more direct attacks against the PPP, as were the two other parameters which multiplied the cost by

a given amount and/or exponentiated it.

Clark and Jacob also made various other suggestions and observations about the problem. For instance, changing any element of $V_{\text{candidate}}$ caused all the entries in the U computed from it to change by ± 2. From this they worked out that it was possible to tell whether the number of entry flips required to achieve $V_{\text{candidate}} = V$ was odd or even. They also suggested exploiting the majority vector, and aspects of Knudsen and Meier's methods.

The attack with timing support The attack as so far described had already comprehensively broken the PPP. The (101, 117) version could now be solved in less than 2^{54} time. (151, 167) could be solved in less than 2^{60} time. Clark and Jacob had attacked (131, 147) instead of (121, 137); this had been solved fastest of all. We confess that this leaves us slightly unsure about the status of the (121, 137) version of the problem, but given the results against the supposedly hardest feasible parameters it would be foolish to assume that this version could provide meaningful security. (We note also that Clark and Jacob had used the same techniques to solve instances of the PP of size (201, 217) and (401, 417) with a 100% success rate, and sizes (501, 517), (601, 617) with success rates of 70% and 50% respectively.) The real power of the attack had stemmed from problem warping (In fact, Knudsen and Meier's observing bits that the various runs agreed on had not yet been exploited here!); the timing/thermostatistical aspects merely represented further improvement on top.

As we currently understand it, the timing enhancements have not yet been used in an unaided attack on the PPP. Information on the number of initially-settled bits that proved to be correct for the "best" run/cost-function pair for each of the problem instances attacked, together with the corresponding number of bits the annealing stage had correctly deduced before terminating, is given, and there is some discussion of how such results could reduce the complexity of the brute-force stage. It looks as if more experimentation would be needed to decide how many initially-settled bits one should assume correct in a practical attack, but the approach would appear to be the same; over all ($|\text{cost functions}| \times |\text{runs per cost function}|$) runs, the best solutions will yield results, for the worst ones we give up before too much time is spent. Other ways to exploit this by brute-force on *pairs* of runs are suggested; however these do not improve on the results obtained by problem warping.

Combining timing support with aspects of Knudsen/Meier's attack (perhaps accepting

the agreement of most, not necessarily all, runs on certain secret vector entries), and ranking the entries in terms of how quickly on average each became "stuck" (the higher the rank, the more likely the entry was to have become stuck at the correct value) were suggested as ways to obtain further improvements.

Other subsequent work After Clark and Jacob had demonstrated the insecurity of even the largest of the suggested parameter sizes, the PPP no longer attracted interest as a basis for identification schemes. Parameter sizes sufficient to render their attacks impractical would result in protocols noticeably less efficient than those relying on the difficulty of other problems, and even then it would be hard to predict the effects of new cryptanalytic techniques (or indeed refinements to the existing techniques). However, we wish to mention, briefly, two subsequent papers on the subject.

In the first of these [224], Uddin and Youssef apply ant colony optimisation to the problem instead of simulated annealing. Of some interest is their statement that minor variations to the cost function parameters are necessary during the search because of "our experimental observation that, for a given cost function, some bits in the solution have a tendency to get stuck at a wrong value throughout the search process". They do not specify how many bits would become fixed at the correct values before this happens, so we cannot compare this to the seemingly very different behaviour of the SA algorithms. We do note, however, that the sizes of the matrices A attacked do not correspond to Pointcheval's recommendations; in fact they do not always guarantee a non-invertible A, and it is possible that differences in the behaviour of the search algorithms could be partly attributable to this.

The second of these [206] we have in fact already mentioned. This paper, published in 2003 and coauthored by Pointcheval, focuses mainly on republishing existing material and on the protocols and their implementation, as opposed to new research on the underlying problems. However, it does contain some interesting material that contributes to a better understanding of the Permuted Perceptron Problem - for instance, until its publication, Pointcheval's proofs that PP, PPP, and approximation of the PP were NP-complete had not been translated from French into English.

3.6.3 Metaheuristic attacks on block ciphers

Attempts have also been made to apply metaheuristics to the cryptanalysis of modern-day symmetric ciphers. Most of this research has focused on block ciphers, although there has been some work on stream ciphers [77]. Most of it has also been carried out by members of the artificial intelligence community; the cryptographic community not appearing to have shown much interest in these methods.

The Tiny Encryption Algorithm (TEA)

TEA [193] is a Feistel cipher with a 128-bit key size and 64-bit block size. It was introduced by Needham and Wheeler in 1994 with the intention of providing an efficient algorithm that could easily be implemented on as wide as possible a range of hardware architectures and programming languages, while still providing a high level of security. A large amount of this security came from its having sixty-four rounds; which it was able to afford due to the efficiency of its round function.

Certain aspects of TEA's design are quite notable. First of all, it does not use any form of table lookup (such as S-boxes), considering this to be too slow and potentially complicated (possibly on some architectures more than others). Instead, the entire cipher is specified in terms of XOR, logical bit-shifts, and arithmetic addition. Bearing in mind the rationale for similar design decisions in the stream cipher Salsa20 [21, 22] the choice of these operations may also have provided some degree of security against timing attacks.

Secondly, every two rounds TEA's round function changes. The $2i$th and $(2i-1)$st round functions add i times the value $\delta \approx 2^{31}(\sqrt{5}-1)$ to a copy of the round input, depending on the modular reduction caused by this value overflowing its allocated storage and on the irrationality of $\sqrt{5}$ to add randomness. (The $\approx$ comes from the fact that no representation of this number using a finite bitstring can be completely exact.) TEA relies entirely on this fact to provide security against slide attacks, since its key schedule repeats every two rounds! Each pair (round $2i-1$, round $2i$) of TEA rounds using different round keys and the same value of δ is referred to as a "cycle".

Various weaknesses were discovered in TEA after its publication. At Crypto '96, Kelsey, Schneier, and Wagner presented a paper [152] in which they showed that every key encrypted plaintexts to the same ciphertexts as three other keys. They also showed how to obtain these three other keys. While this did not make much difference to TEA's security as an encryption algorithm (its effective key strength was reduced from 128-bit

to 126-bit), it rendered it utterly insecure as a basis for cryptographic hash functions, as the operation of TEA on a (plaintext, key) pair did not possess either second-preimage resistance or collision resistance.

Kelsey, Schneier and Wagner uncovered further weaknesses in the algorithm, published in 1997 [153], motivating TEA's creators to design a replacement algorithm, XTEA [194]. (The reader may note that XTEA was published before this set of weaknesses were; this is because Needham and Wheeler had been informed of them by Wagner prior to the ICICS conference.) An even stronger algorithm, XXTEA, was published by Needham and Wheeler in 1998 [195].

Bitmask-based attacks against TEA. In two papers published in 2002, and 2003, Hernández-Castro et al. proposed a method of using genetic algorithms in a distinguishing attack on reduced-round TEA (not XTEA or XXTEA). Their method [139, 138] worked as follows:

1. The initial population consists of a set of one hundred 192-bit strings (bitmasks). It is not explicitly stated how these are generated, we assume randomly. 2^{11} random (plaintext, key) pairs are generated at random.

2. For each string in the population:

   (a) A bitwise AND is carried out with this string and each of the (plaintext, key) pairs. That is, each input pair has its bits set to zero where the mask-bits are zero.

   (b) The thus-altered input pairs are run through the reduced-round cipher, and the resultant ciphertexts subjected to statistical tests for randomness.

   (c) The fitness function assigns a score to the bitmask based on its performance, such as the χ^2 value. The higher the value, the less random the output. If the bitmask's performance is sufficiently high, a higher-valued function of the weight of the bitmask is assigned as a score instead, to give this bitmask a significant advantage.

3. Selection, crossover, and mutation are applied, and either the next generation of tests occurs or the best candidate (the one with the largest Hamming weight) is selected.

The bitstrings evolved were intended to define the distinguishing attacks; up to 2^{11} random (key, plaintext) pairs would have bits set to zero in accordance with the bitmask, be run through the function, and tested for non-randomness as before. If the outputs were judged to have a sufficiently high score for non-randomness, the algorithm would decide that the function was TEA and not a random function.

(No method for turning this into a key extraction attack is specified. The intuitive approach would be to attack a black box implementing either TEA with key k, or a random function, using a 64-bit mask on the plaintexts. This would require a bitmask which did not impose constraints on the key bits to have been generated, so in practice it would seem some other way to exploit the non-randomness would be needed.)

Direct searches for the encryption key We have already mentioned the folly of a direct search for the encryption key, although this has nevertheless been attempted in various papers [19, 20] [190]. One particular example which we wish to look at is the work of Song, Zhang, Meng, and Wang [181], in which they attack four-round DES in this way using a genetic algorithm.

The attack assumes that a quantity of known plaintext/ciphertext pairs are available to the cryptanalyst. The candidate keys are used to encrypt the plaintexts, and the fitness function is based on how many bits the results agree with the correct ciphertext in. (It is not clear how many plaintexts each key is tested on) At the end of the search, the "optimum" members of the final population (those members which have achieved fitness levels above a certain threshold, approximately 80%) are identified. Any key bit which a of these have agreed upon (The value of a varies, but the average value is 67.5%) is set to that value in a new seeding population, with which the search is repeated.

This should be familiar from our earlier discussion of the PPP. In particular, as well as the possibility that something analogous to a timing channel could be used, the reader will realise that this must set several bits to incorrect values. If these bits cannot be altered during the later searches, then no method has been provided to identify and correct these - and even if they can change, it is not guaranteed that mistakes will be corrected during the subsequent search. This is not addressed by Song et al. - their paper mentions the number of correct bits after the first search (up to 13), but does not mention the number of incorrect bits. They claim that they can continue the search to find the remaining key bits and complete the key, but we are not sure. In any case, this approach could not break DES reduced to more than a handful of rounds.

Other applications of metaheuristics to cryptanalysis

A minority of researchers have begun to recognise other ways in which metaheuristics (and, we suspect, search algorithms which are not metaheuristic in nature) can be used in cryptanalysis. There are three particular directions of research, all of which use metaheuristics as a tool to optimise existing cryptanalytic techniques. This is a common thread - realising that metaheuristics must be targeted at aspects of the cipher or its cryptanalysis where there is reason to believe sufficient non-randomness exists to create a searchable fitness landscape, and that such non-randomness in the cipher is usually exploited by, or brought to light during, cryptanalysis.

Searching for differential characteristics In a paper published in 2004 [8], Bafghi and Sadeghiyan use weighted graphs instead of tables to represent the difference distribution tables of Serpent's S-boxes. The S-boxes are linked together by further edges corresponding to the paths the data follows through the cipher, and an ant colony algorithm is used to find good differential characteristics, with the ants traversing the edges of the graphs trying to find the paths with the maximum weight (corresponding to the maximal likelihood of the differential characteristic occurring).

Serpent was considered the most secure of all the submissions to the AES contest, more secure than the AES itself, and is hence one of the most important ciphers for cryptanalysts to focus on. As a result, it has been shown to be highly secure against differential cryptanalysis. This does not, however, mean that searching for differential characteristics for it is a cryptographic dead end - ciphers designed to resist differential and linear cryptanalysis have been comprehensively broken using variants of these techniques, such as the boomerang attack [229] discovered by Wagner in 1999.

The boomerang attack is able to attack ciphers using a pair of good short differential characteristics, instead of one long differential characteristic. Let the function defined by the cipher be denoted E. The attack uses chosen plaintext *and* adaptive chosen ciphertext as follows: $(P, P' = P \oplus \Delta)$ is input to the cipher. After the first x rounds of the cipher (call them E_0), we expect with some probability that $E_0(P) \oplus E_0(P') = \Delta^*$ for some value Δ^*. We do not necessarily expect to observe any value of $E(P) \oplus E(P')$ with sufficient probability to use in a conventional differential attack. Let the function defined by the remaining rounds of the cipher be denoted E_1. We need to have a differential characteristic for E_1 whereby $E_1(R) \oplus E_1(R \oplus \nabla^*) = \nabla$ for some (∇^*, ∇) with reasonable probability.

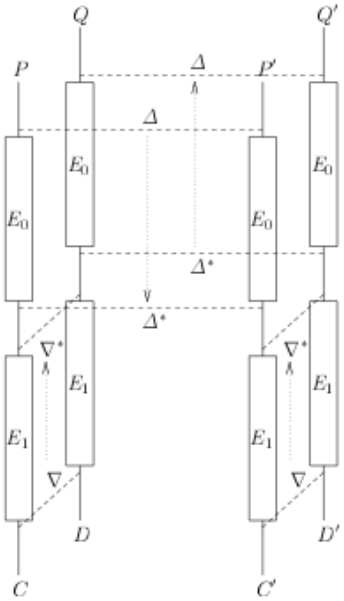

Figure 3.10: Diagram illustrating the boomerang attack [229].

Let $D = E(P) \oplus \nabla$. Let $D' = E(P') \oplus \nabla$. We request the decryptions of D and D'. If the resultant plaintexts have difference Δ, we use the quartet $(P, P', E^{-1}(D), E^{-1}(D'))$ in the second stage of our attack, which includes the testing of candidate TPS values.

Boomerang attacks evolved into "amplified boomerang" attacks [150] using chosen-plaintext quartets instead of chosen plaintexts and ciphertexts, before evolving again into the *rectangle* attack [28] against Serpent reduced to ten rounds (from thirty-two).

The rectangle attack utilised a characteristic for the first four rounds of Serpent with probability 2^{-29}, and one for rounds 5-8 with probability 2^{-47}. (Serpent, like TEA, varies its round function throughout the cipher, albeit with periodicity eight. The last round also has less diffusion than the others.) From these, characteristics for rounds 5-9 with probability 2^{-60}, and rounds 4-9 with probability 2^{-93} were discovered.

Bafghi and Sadeghiyan's work was carried out in the context of this work, and two other papers which had achieved the best-known characteristics for various sequences of rounds of Serpent [151] [62]. Using the ant colony technique, they were able to find a characteristic for rounds 1-5 with probability 2^{-65} instead of the 2^{-67} reported by Chan et al. [62], and a characteristic for rounds 1-6 with probability 2^{-94} instead of Chan

et al.'s 2^{-97}. Their algorithm also succeeded in finding a characteristic for rounds 1-4 with the same probability as that found by Biham et al. [28]. In a later paper [9], coauthored with Reza Safabakhsh, neural networks did not improve on these results, however "Boltzmann" neural networks incorporating ideas from simulated annealing were able to obtain a characteristic for rounds 1-7 with probability 2^{-125}. Whether any of these will be used in attacks, however, remains to be seen.

Improved TPS evaluation in differential cryptanalysis In the earlier section on differential cryptanalysis, we mentioned local maxima occurring when incorrect keys nonetheless caused the correct output difference to occur unexpectedly often, and wondered whether these might be usable in metaheuristic attacks. Albassal and Wahdan [2, 3, 4] start with a similar idea and analyse it in more depth.

The key observation they make is that for TPS candidates with several bits in common with the correct subkey, the expected output difference will occur more often than expected, (although not as often as for the correct subkey) and that parts of the output difference will occur with the expected frequency. The reason for this is that these will have the correct values for some, though not all, of the S-boxes involved in the final round of the characteristic, meaning that they will yield the correct values for the corresponding bits of the output difference of these as often as the correct key will. Since the chance of obtaining the correct output difference for, say, one random S-box with the wrong key is higher than that of achieving the correct output difference for several S-boxes with the wrong key, they will also result in the complete output difference more often than expected. An examination of the frequency counts from a differential attack on a toy cipher [140] does indeed demonstrate this phenomenon occurring. The result is that the success rates of the various TPS candidates create a searchable fitness landscape, which can potentially be exploited to reduce the number of chosen plaintexts required.

In their first paper on the subject [2], Albassal and Wahdan exploit this in an attack against the aforementioned toy cipher, before going on to attack a simple Feistel cipher in a follow-up paper [3]. Genetic algorithms are used in both cases, although we would have liked to have seen hill-climbing or simulated annealing tried. Another paper by the same authors [4] describes an attack on another simplified Feistel cipher using neural networks, and presents a useful introduction to these, but does not really offer anything more than the other papers in cryptographic terms.

We found the second of these papers to be the most interesting; since the S-box used

in the Feistel network was a row from one of the DES S-boxes, the question of how to improve on this attack with the intention of attacking DES was implicitly raised. We note the following points:

- The fitness function rewards candidate TPSes based on how many times each yields the correct output difference. For the Feistel cipher as described [3], we think that rewarding them based on how many S-boxes they cause to exhibit the correct difference for each chosen plaintext pair would be more suitable, as this seems to better reflect the behaviour of the nearby keys. Or rewarding based primarily on this, and secondarily on occurrences of the full correct output difference.

- Whether this would apply to DES itself would need further investigation. An incorrect key bit input to a final-round S-box could, because of the DES expansion, cause two adjacent S-boxes to exhibit the wrong output difference. We could, however, find ways to exploit a tendency for certain solution candidates to frequently get two S-boxes wrong when candidates different in only one bit do not...

- When a TPS gets several, but not all, bits of the output difference corresponding to S-boxes right, it may be worth rewarding/penalising those for which it was incorrect according to the DES design criteria. So, for example, an S-box in which the output difference was wrong in only one bit should be penalised more than one for which it was wrong in two, as a one-bit input difference to a DES S-box cannot result in a one-bit output difference.

- This technique would appear not to be so easily applicable to linear cryptanalysis. In differential cryptanalysis, a partial success in the decryption of one of the ciphertexts can be evaluated and result from a partially correct key; this is not the case for linear. However, in the case of DES, we might in some cases expect keys which are only wrong by one bit to achieve higher success rates than other keys; since such keys must cause an S-box output different in 2 or more bits to that for the correct key, the possibility that the parity of the deciphered bits will be the same as for the correct key will be increased.

  That said, the best-known linear approximations for DES [180], although they involve approximations involving all the S5 output bits, do not have the corresponding key bits as part of the TPS and thus cannot exploit this phenomenon. But the output bits when key bits for S5 are counted on still have a parity that we can predict

the average behaviour of when one key bit is incorrect. It depends on which input bit is incorrect, but the parity stays the same more often than it flips. So there may still be something in this that we can exploit.

How much this can improve the complexity of the differential attack by is unclear. The authors claim that the attack complexity can be halved, but the reasoning appears tenuous and may not hold for a more complicated cipher; however by making improvements to the fitness function as we have suggested the genetic attack's complexity could be improved further.

Apparently building on Albassal and Wahdan's results, Itaim and Riff claim [142] to use the same methodology to attack reduced-round XTEA. However, they do not state the differential used, the differential characteristic used, or the signal-to-noise ratio; so we cannot verify this claim.

In addition, although they claim to improve upon the complexity of a traditional differential attack, in almost none of the cases presented has their final solution been correct for all thirty-two TPS bits. The complexity faced in a real-world situation by a cryptanalyst who did not know the correct key of

- trying to work out how many key bits were correct, and
- trying to find out which ones were incorrect

is ignored.

Searching for nonlinear approximations to S-boxes The existence of linear cryptanalysis has led to several researchers attempting to generalise it thus: since the attack relies on linear functions approximating the actions of S-boxes, why not nonlinear functions on some subset of the S-box's input - or indeed output - bits?

There are various reasons for this [130], in particular that it is not as easy to link the approximations to successive rounds together as it is for linear cryptanalysis. In the case of Feistel ciphers, nonlinear approximations are in most cases *impossible* to link up between rounds in the same way as linear approximations [164]! More recently, Courtois demonstrated [104] that bi-linear functions multiplying together linear expressions of the left and right-hand data blocks of Feistel ciphers could be linked together and used in cryptanalysis. We do not study these here, although Courtois has identified finding the

best bi-linear approximation to a cipher as a hard problem and we very much hope to apply metaheuristics to it at some later stage.

Instead, we consider the use of metaheuristics in the light of Knudsen and Robshaw's work [164], in which they noted that nonlinear approximations to the first and last rounds of an overall cipher approximation could be made to work, as long as the only nonlinear expressions were of the input/output bits of the overall approximation. We focus on the work of Clark, Tapiador, and Hernández-Castro in finding nonlinear approximations of this sort to S-boxes in the first/last rounds of a linear approximation [78]. In this work, they focused on the unusually large (9 input bits, 32 output bits) S-box of the block cipher MARS [49], which was one of the finalists during the contest to determine the Advanced Encryption Standard.

Although the methods used in the paper are trivially adaptable to the S-boxes in the final round of a linear approximation, they have only been used to find approximations for the first round. This means that a non-linear function of the S-box input bits is equal to a linear function of the output bits with some probability bias, and the aim is to maximise this bias.

Now, at this point, we should make certain facts clear. Although this research has obtained surprising results against the MARS S-box using simulated annealing, there does not seem to have been adequate reason to believe that this would work at all in the first place, and it is not clear why it did!

Let us be more specific. Clark et al. applied simulated annealing as follows:

- The set J was defined as the set of S-box input bits on which the nonlinear function f operated. A function f_j determined which of the S-box input bits corresponded to each of f's input bits.
- A value, $\hat{n}$, was set at the start determining the size of J. This variable did not change during the annealing process. Neither was it ever set as high as 9, the number of input bits to the MARS S-box.
- The non-linear function f was represented by its truth table.
- The linear function g of the S-box output bits was represented by a bitmask.

At each step, the candidate solution had to move to a neighbouring candidate. Parameters were specified detailing the probability, at each stage, of its doing so by each of

these methods:

- varying the truth-table representation of f;
- varying the bitmask representation of g, or;
- changing f_j (and hence J).

The representation of f using its truth table appears to have been motivated by the fact that small changes to a Boolean function's truth table result only in small changes to cryptographically relevant properties such as its nonlinearity and autocorrelation. This fact had been, as we have already stated, the motivation behind the use of metaheuristics to evolve cryptographic functions. However, there is no existing research on the effect of truth table changes on the closeness of a random Boolean function f to a linear function g on a different set of variables. (or, more precisely, a set of variables related to f's variables and other variables by the actions of a highly nonlinear high-degree mapping.)

The behaviour of the search was, nevertheless, surprising. For nonlinear functions on a large number of the S-box input bits, after a fairly modest initial improvement in the solution quality, the algorithm would fail to significantly improve on the solution quality for approximately 500,000 iterations. After this it would start obtaining dramatic improvements, and continue to do so for between 400,000 and 600,000 more iterations [78], before reaching a maximum and failing to improve further.

The best linear approximation for the MARS S-box has been shown to have a bias of 84/512. The best nonlinear approximation obtained by Clark et al. was for the case $\hat{n} = 8$, with a bias of magnitude $151/512 \approx 0.295$. Nonlinear approximations, as observed by Knudsen and Robshaw [164], often exist with higher biases than linear approximations, and for $\hat{n} \geq 6$ simulated annealing almost always found nonlinear approximations with better bias than the best known linear approximation.

While this was the first time simulated annealing had been used against a block cipher in this fashion, the same team had also tried to find approximations to reduced-round versions of the stream cipher Salsa20 [77]. The results, published two months earlier, were not so impressive as the results against MARS - in particular, assumptions had had to be made about the values of the key and a related value known as the "initialisation vector", and it was not clear how the results obtained would apply when these assumptions did not hold. Nevertheless, the annealing process had shown almost identical behaviour, modest

initial improvements transforming into a long period of significant improvements before tailing off.

While suggestions to improve on this research have been suggested by its authors, consisting mainly of speculations as to the effect of different representations of f and g, it seems to us that the two most obvious starting points for further research are:

1. find out why this worked as well as it did, and
2. find out if it can be applied to the S-boxes of other ciphers

Most of our own research into evolving nonlinear approximations is presented in Chapter 6, in which, among other things, we address the first of these issues in some detail, and, through experimentation, are able to answer the question "is this applicable to the S-boxes of other ciphers" with a resounding YES!

Part of the reason that we viewed it as necessary to look at ciphers other than MARS is that MARS has not been widely used since its introduction in 1999. Other AES finalists, such as Serpent and Twofish (and AES itself!) were viewed as far superior - Serpent in particular being considered the most secure of the AES finalists, and in recent years there have also been promising new block cipher designs intended for use in low resource environments such as RFID tags and smart cards.

- Serpent uses 4×4 S-boxes, and as there are $2^{4+2^4} = 2^{20} = 1048576$ nonlinear approximations of this sort to a 4×4 first-round S-box, we do not expect metaheuristics to offer any improvement over exhaustive search. However, neither metaheuristics nor exhaustive search have yet been applied to this problem, and since the best current cryptanalysis of reduced-round Serpent utilises a variant of linear cryptanalysis [196], it seems to us worthwhile to search for nonlinear approximations which may improve on this attack.

- Twofish's 8×8 S-boxes are derived from two fixed mappings in a way dependent on part of the key. These are more complicated than conventional fixed S-boxes. We note that Twofish's designers considered average probabilities over all possible S-box key inputs when analysing the cipher's resistance to differential cryptanalysis, but are not yet familiar enough with this sort of S-box to reach any conclusions regarding the applicability of metaheuristics.

- For a DES S-box, there are $2^{4+2^6} = 2^{68}$ such approximations if the nonlinear component is in terms of the input bits, and $2^{6+2^4} = 2^{22}$ if it is in terms of the output bits. We note that due to DES's Feistel structure, the former case is applicable both in the first and last rounds

  DES uses eight different S-boxes in parallel, so to examine them all requires 2^3 separate searches of a search space 2^{68} in size. We are not sure how easily the computational resources for such an exhaustive search could be brought together; even if this is feasible it may be beyond institutional budgets or the interest levels of distributed computing projects to do so, or it may be felt that it would take too long. The DES S-boxes therefore represent good candidates for metaheuristic analysis in this fashion, especially since Triple DES is still in use.

- For an 8×8 AES S-box, there are $2^{8+2^8} = 2^{264}$ first-round approximations of this sort; likewise the last round. This is well out of the reach of exhaustive search and AES would therefore seem to be an excellent target! However, due in no small part to the "wide trail" design strategy, no significant linear cryptanalytic attacks against reduced-round AES currently exist, and finding linear approximations to several rounds of the cipher which could be combined with nonlinear approximations in a viable attack may yet prove intractable.

It should be noted that even when an S-box can be exhaustively searched for the "best" approximation, the possible need to repeat the searches to find approximations better suited to overall cipher approximations, even if they have a lower bias than that S-box's "best" approximation, and the issue of storage for large numbers of approximations, would appear to make a means of quickly searching for a nonlinear approximation under specified constraints, such as metaheuristics, an attractive prospect! We discuss the results of our search for nonlinear approximations to the AES, Serpent, DES and other S-boxes in Chapter 6, as well as our investigation into why simulated annealing was so effective, and build further on this research to define a nonlinear attack against eleven rounds of Serpent. In particular, the bias of a single nonlinear approximation does not turn out to be enough to assess its cryptanalytic usefulness, and we explain why the cryptanalyst must in most cases evaluate the biases of several related nonlinear approximations as well.

3.6.4 An attack with a quasi-metaheuristic methodology.

In 2002, three physicists [149] published a proposal for a new key-exchange protocol that would not rely on number theory. It would instead rely on a phenomenon from chaos theory known as *chaotic synchronization*, and on so-called "artificial neural networks" (ANNs). Just as most metaheuristics are nature-inspired search algorithms, artificial neural networks may be viewed as nature-inspired machine-learning methods - the inspiration from nature in this case being the human brain!

Alice and Bob would both possess a secret ANN, part of which would be defined by a $K \times N$ matrix of "weight" values $w_{k,n}$ $((-L \leq w_{k,n} \leq L)$ for some value L, $(1 \leq k \leq K)$, $(1 \leq n \leq N))$. Apart from the values in the weight matrices, the ANNs would be otherwise identical. Recommended parameter values were $(L = 3,\ N = 101,\ K = 3)$. In addition, at the start of each round, a $K \times N$ matrix X such that all matrix entries $x_{k,n} \in \{-1, 1\}$ would be generated at random and in public.

The values $x_{k,n}$ would act as input to the two ANNS. A set of K individual components - or "perceptrons" - of each ANN would output values o_k, each of which would be input to higher-level parts of the ANN. The ANN would eventually output a single value $O \in \{-1, 1\}$. Let O_A denote the output of Alice's network, O_B the output of Bob's network.

If the two values differed, the two parties would simply move on to the next round of the protocol, starting with recalculating the matrix X. If they were the same, the parties would carry out an "update" procedure on the weights of any perceptrons whose outputs had agreed with the network's overall output. Each perceptron could be viewed as corresponding to a different row in the matrix of weights.

Over time, this would lead to the two weight matrices becoming identical, and outputting identical values in every round after this had occurred. After some predetermined number of rounds (20-30 being recommended; a figure we personally consider too low) in which the networks had constantly output identical values, it would be presumed that this had happened - that the networks were "synchronised". Alice and Bob's shared cryptographic key would then be derived by computing a one-way hash of the contents of the weight matrix.

This protocol appears to be far less efficient than, say, the Diffie-Hellman protocol. It requires a large number of rounds, and in each of these rounds the $K \times N$ matrix of public data must be recalculated from scratch. Moreover, there is a slim probability that the matrices will be deemed to have synchronized when in fact they have not, and the

networks will have been outputting identical values by sheer coincidence - a coincidence that might occur all too often if the protocol were implemented in billions of computers worldwide! Presumably the motivation was to try to derive a protocol that would not be vulnerable to attacks using quantum computers and Shor's algorithm.

Klimov, Mityagin and Shamir [158] identified weaknesses in this protocol, and derived three different attacks against it. One of these attacks, though not in fact a genetic algorithm, was referred to as one, and utilised several concepts from GAs.

The attacker has a population of candidate solutions, each of which is an ANN defined as above. At the start, the weight matrices for these are computed at random. The size of the population will vary throughout the attack, which is conducted in parallel with the protocol. There is no guidance on its original size, however a "threshold" size M is defined which will be relevant during the attack. The aim of the attack is to synchronize one of the candidate solutions with Alice's ANN before Bob's ANN synchronizes with it.

In each round of the protocol, if $O_A \neq O_B$, Alice and Bob move on to the next round of the protocol, and the attacker does nothing.

If $O_A = O_B$, the attacker's pseudo-GA proceeds in one of two ways depending on the size of the population. If the population contains less than M members, then for the recommended value $K = 3$, there are four different sets of o_k which could have resulted in output O_A. Instead of a crossover operation, each member C of the population reproduces asexually, creating four copies (C_1, C_2, C_3, C_4) of itself. Each C_i is then subjected to what the update procedure would have been had its perceptrons actually output the ith set of o_k values - this being analogous to the mutation phase.

(The attacker still needs to keep a record of the outputs of the networks during this, for reasons that will soon become clear. It is also necessary to keep track of the parent neural network of each of the new networks.)

If there are M or more members in the population when it is observed that $O_A = O_B$, a phase akin to the selection phase begins. All the ANNs in the population compute their outputs, and any which output a value other than O_A are deleted. The remaining networks are then updated using their actual values of o_k.

Using $M = 2500$, Shamir et al. observed a greater-than-50% chance that one of the members of the population would synchronize with Alice's network before Bob's did, and survive, updating in step with Alice's network, for long enough (presumably the same "20-30" generations) for the attacker to observe that one of the networks was outputting

the value O_A constantly and conclude that it had synchronized.

3.6.5 Attacks using non-metaheuristic search algorithms.

Other researchers have expressed problems in cryptanalysis as search problems, before applying general-purpose, non-deterministic search algorithms to these. Of particular note are:

- Shamir's use of *integer programming* algorithms to break the so-called "single-iteration knapsack cryptosystem" [214]. The integer programming algorithm used was later superseded by the "LLL" algorithm [174], which presumably would have allowed the attack to succeed even more efficiently.
- Borghoff et al. [41] expressed the stream cipher Bivium as a "mixed-integer programming" (MIP) problem, allowing them to attack it using linear programming algorithms.

Chapter 4

Evolving balanced Boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree, and very high nonlinearity.

Previous work on evolving Boolean functions focused on the combiner model for LFSR-based stream ciphers, and did not take resistance to algebraic attacks into account. We give justification for focusing on the filter model instead. We demonstrate in this chapter that smooth search landscapes exist for the properties of Boolean functions governing resistance to algebraic attacks, and evolve Boolean functions with optimal resistance to these, and with the high nonlinearity and algebraic degree that are also necessary. Raising the computational resources allocated to the tasks, we improve on the best-known theoretical results in the literature.

4.1 Introduction

Combiner and filter functions for shift register-based stream ciphers need to satisfy various cryptographic criteria. They must be *balanced* [52], possess high *nonlinearity* to resist fast

correlation attacks [55, 126, 50, 221], and must possess high *algebraic degree*:

- to resist the RØnjom-Helleseth attack [132],
- to resist the Berlekamp-Massey attack [52],
- as a necessary but not sufficient condition to resist fast algebraic attacks [103].

(Furthermore, where n denotes the number of input bits of the Boolean function, an algebraic degree less than $\lceil \frac{n}{2} \rceil$ restricts the degree of so-called *algebraic immunity* that can be achieved.)

In the case of combiner functions, a high order of *correlation immunity* is also necessary [216, 217]. For filter functions, correlation immunity of order 1 is considered sufficient [55]. Unfortunately, the criteria of correlation immunity and algebraic degree are in conflict with one another, and the higher the correlation immunity of f, the lower the value that can be achieved for its degree - which increases the desirability of a model relying on filter instead of combiner functions.

(Correlation immunity of order 1 for a filter function is typically achieved by generating a function g which is as close as possible to the optimum for the other desirable criteria, and then using a shift register state bit x_{n+1} which is not input to g to define a function $f(x_1, \ldots, x_{n+1}) = g(x_1, \ldots, x_n) \oplus x_{n+1}$. It may be necessary to apply an affine transformation to the input bits of f and/or g [52, 51].)

Correlation immunity of order m for a balanced function is also referred to as order-m *resiliency*.

Until the early 21st century, these were the only criteria which a stream cipher's filtering/ combining function needed to satisfy. However, the discovery by Courtois et al. of algebraic attacks [109] and fast algebraic attacks [103] changed this, forcing:

1. An increase in the number of input bits needed by these functions (from "about 10" to "at least 13 and in practice much more, maybe 20" [55].) Where the shift register is a 256-bit LFSR, Braeken et al. [43] show that a balanced filter function on at least 14 input bits is needed to keep the time complexity of fast algebraic attacks below 2^{80}. The "RØnjom-Helleseth attack" [132] is a more recent form of algebraic attack, and it has been claimed [203] that filter functions need more than 30 input bits to resist it.

2. The introduction of two new criteria to quantify the resistance of Boolean functions f to these attacks. The first of these is the aforementioned *algebraic immunity* (AI) [57]. The second criterion, which involves "$(d_g, d_h)-$relations", is unnamed at present; we shall refer to it as *fast algebraic resistance*. A criterion known as *fast algebraic immunity* attempts to unify the two; however we consider it to be flawed.

In a 2007 paper by Fischer and Meier [125], algebraic attacks on "augmented functions" are discussed. For a given filter function f, shift register update polynomial L, and integer $m > 1$, the augmented function S_m is a vectorial Boolean function which takes as input the internal state of the shift register and is defined as the concatenation $(f(x)|f(L(x))|f(L^2(x))|\ldots|f(L^m(x)))$. Since the properties of augmented functions depend on both the shift register update function and the filter function, and since this chapter only studies filter functions, it is beyond our remit to examine resistance to such attacks here.

Finding Boolean functions which combine optimal or near-optimal resistance to algebraic and fast algebraic attacks with the various other desirable properties has proven difficult. The need for high algebraic degree has led some researchers in this field to abandon the combiner model and focus entirely on filter functions; for which only a few suitable constructions have been found.

The first such was the Carlet-Feng construction [55]. This defines a class of balanced Boolean functions on n variables with algebraic degree $(n-1)$ and algebraic immunity $\lceil \frac{n}{2} \rceil$ (These are the optimal values of AI and degree for balanced f). A lower bound exists on their nonlinearity; this lower bound is not near-optimal but in practice the nonlinearity of the functions in this class was observed to exceed $2^{n-1} - 2^{\lfloor \frac{n}{2} \rfloor + 1}$ for $n \leq 11$. Furthermore, the fast algebraic resistance of the constructed functions was examined and, for functions with less than 10 variables, shown by experimentation to be optimal.

The Carlet-Feng class was independently rediscovered by Wang et al. [148], with a slight increase in the lower bound for nonlinearity. Functions with higher nonlinearity than previously achieved for $n = 8$, $n = 9$ and $n = 16$ were also presented.

Carlet [54] demonstrated that the two constructions were the same. He also stated that the functions could be implemented without needing to store a lookup table in memory; the computation of the Boolean function could be reduced to calculating a discrete logarithm, which, he stated, was feasible using the Pohlig-Hellman algorithm [134] when the function operated on 20 input bits or less. He stated, however, that this was the highest value of n

used by such functions, and if we accept Pasalic's claim [203] that more than thirty are in fact needed, then it is not clear whether such an implementation is in fact viable for these larger functions.

The construction was improved upon in two later works coauthored by Carlet [56, 58]. The constructions in these papers obtained increased nonlinearity for $n = 10$, and results were obtained for much larger values of n than the previous papers had dealt with. In addition, the lower bounds on nonlinearity for the Carlet-Feng functions were tightened, and another balanced construction was presented which improved on the nonlinearity for some values of n but not others.

It is notable that the only apparent way to compute the functions in one of these papers [56] without needing a lookup table also involves calculating a discrete logarithm, and this would appear also to be a necessary step in computing the functions described in the other paper [58]. We reiterate that it is not clear whether this is viable for functions on more than 20, or indeed more than 30, input bits; thus motivating a search for alternatives.

In this chapter, we apply simulated annealing to the problem of finding balanced Boolean functions with optimal AI, FAR, and algebraic degree, and high nonlinearity. As stated in Section 3.5, this technique has achieved success in a similar context, when searching for combiner functions with high nonlinearity, low autocorrelation, and good tradeoffs between high degree and high order of correlation immunity [80, 84, 87]. In particular, functions with profiles not hitherto obtained by any construction were found [80, 84] using this method, and since combining optimal algebraic immunity and fast algebraic resistance with the other cryptographically desirable criteria for filter functions remains an active area of research, in which near-optimal values for nonlinearity in particular have not been achieved, we hope to replicate this success here.

(Autocorrelation was believed when first defined [232] to be a potential weakness that might lead to attacks on stream ciphers akin to differential cryptanalysis of block ciphers. As this has not subsequently proven to be the case, and as no eSTREAM finalist other than Dragon [117] was designed in a way that took autocorrelation into account, we will not focus on it here.)

Crucial to the method's success was a technique known as "two-stage optimisation". This technique would use one cost/fitness function for the simulated annealing, and would then hill-climb the results using a different cost function. The idea was that the first cost function would guide the annealing into a region of the search space (here the set of all

Boolean functions of the pertinent size) in which candidate solutions (evolved Boolean functions) of above-average quality as defined by the second cost function were likely to exist. The second cost function would search this region for one of these high quality solutions, and return it. In Clark et al.'s experiments [80], this achieved far more favourable results using two-stage simulated annealing than any previous attempts to construct Boolean functions with metaheuristics.

This chapter is structured as follows: Section 4.2 will provide precise definitions of the various cryptanalytic criteria and the tradeoffs between them. In Section 4.3, we will describe the search landscape defined by the various properties, and justify our decision to represent the candidate functions using truth tables. We will also discuss the various cost functions used. In Section 4.4, we will compare the best Boolean functions found by our search to the best found by means of construction, and discuss avenues for future research.

4.2 Preliminaries

A balanced function is a Boolean function with an equal number of 1s and 0s in its truth table. We are not interested in filter or combiner functions that are not balanced, and the experiments were designed to ensure that these could not be evolved.

The algebraic normal form (ANF) of a Boolean function is its representation as a multivariate polynomial in which the variables are the values $(x_0, \ldots, x_{n-1})$ of the input bits. This representation is unique, and there exist mappings from truth table to ANF and vice-versa (both with matrix representations).

A linear function has algebraic normal form $a_0x_0 \oplus a_1x_1 \oplus \ldots \oplus a_{n-1}x_{n-1}$ $(a_i \in \{0, 1\})$.

The algebraic degree of a Boolean function is defined thus: Let the Hamming weight of a monomial be defined as the number of variables in it - so, for instance, x_1x_4 has weight 2. The algebraic degree d of a monomial is defined as being equal to its weight, and the algebraic degree of a Boolean function is equal to the algebraic degree of the highest-weight monomial in its ANF.

All balanced functions have algebraic degree $\leq (n-1)$.

The Walsh-Hadamard spectrum of a Boolean function $f \in B_n$ contains information on the correlation between f and the various n-bit linear functions. That is to

say, where ω is an integer between 0 and $2^n - 1$, let $\omega_1\omega_2 \ldots \omega_n$ be the bitstring representation of ω. Then entry ω in the Walsh spectrum is equal to:

$$\hat{F}(\omega) = \sum_{i=0}^{2^n-1} (-1)^{f(i)} \cdot (-1)^{\omega \cdot i}$$

Given a Walsh spectrum $\hat{F}$, the truth table of the original function f can be recovered from $\hat{F}$ using the Inverse Walsh Transform [84]. It is, therefore, a valid alternate representation.

The nonlinearity of f is defined as

$$2^{n-1} - \frac{\max_\omega |\hat{F}(\omega)|}{2}$$

Nonlinearity and algebraic degree are partially in conflict. For functions on an even number of variables, the highest nonlinearity possible is achieved only by *bent* functions, which have degree $\leq n/2$ and cannot be balanced.

The correlation immunity of f is the maximal value m such that $|\hat{F}(\omega)| = 0$ for all ω of Hamming weight $\leq m$ (that is, all ω with m ones or less in their base-2 representations).

Correlation immunity m and algebraic degree d conflict with each other very strongly. For a balanced function, $(m + d) \leq (n - 1)$. For any other function, $(m + d) \leq n$.

The autocorrelation spectrum of f is defined thus. Let ω denote the bitstring representation of an integer between 0 and $2^n - 1$. Then

$$\hat{r}_f(\omega) = \sum_{i=0}^{2^n-1} (-1)^{f(i)} (-1)^{f(i \oplus \omega)}$$

and the autocorrelation spectrum is the sequence $(\hat{r}_f(0), \hat{r}_f(1), \ldots, \hat{r}_f(2^n - 1))$.

There is no inverse transformation allowing the truth table to be recovered from the autocorrelation spectrum.

The autocorrelation of f is the maximum absolute value, $\max_{i \neq 0} |\hat{r}_f(i)|$ in the autocorrelation spectrum.

Algebraic immunity (AI) is defined as the minimum degree of the nonzero functions g such that either $fg = 0$ or $(f \oplus 1)g = 0$. [57]. Such g are known, respectively, as *annihilators* of f or of $(f \oplus 1)$. For this reason, algebraic immunity is sometimes known as "annihilator immunity".

A corollary to Theorem 6.0.1 in the cited paper is that $AI(f) \leq \lceil \frac{n}{2} \rceil$.

Fast algebraic resistance (FAR) is defined thus:

For two values d_g, $(d_h > d_g)$, we say that a $(d_g, d_h)-$relation exists for f if two nonzero functions, g and h, exist such that $fg = h, deg(g) < deg(h), deg(g) = d_g$ and $deg(h) = d_h$.

The fast algebraic resistance of f is the minimum value of $(d_g + d_h)$ for all $(d_g, d_h)-$relations on f. Clearly, since $f{\cdot}1 = f$, this is upper-bounded by $deg(f)$. From our viewpoint, this means that any cost function dealing with fast algebraic resistance also deals to some extent with algebraic degree, since the FAR lower-bounds the degree.

For a given $(d_g + d_h)$, different values of (d_g, d_h) lead to different attack complexities. Various tradeoffs have been discussed [103, 43]; however at present the cipher designer simply aims to achieve a $(d_g + d_h)$ too high for any (d_g, d_h) to lead to an attack, and preferably equal to the maximum value (for a balanced function) of $(d_g + d_h) = (n - 1)$.

It is shown by Braeken et al. [43] that in any $(d_g, d_h)-$relation, d_h is greater than or equal to the algebraic immunity of f.

Fast algebraic immunity (FAI) is an attempt to unify the criteria of algebraic immunity and fast algebraic resistance. It is defined by Johansson and Wang [143] as:

$$FAI(f) = \min\{2{\cdot}AI(f), FAR(f)\}$$

We believe that this criterion is inadequate, and illustrate our reasons as follows: Let $f \in B_{13}$ be a Boolean function with fast algebraic resistance 12. Clearly, the optimal value of $AI(f)$ is 7. However, when $AI(f) = 6$, the value for fast algebraic immunity is the same as if it were 7, since in both cases $FAI(f) = 12$.

4.3 The experiments

4.3.1 Representing candidates as truth tables

So far, we have referred to three possible representations of Boolean functions:

- Their truth tables.
- Their algebraic normal forms.
- Their Walsh-Hadamard spectra.

An additional representation in the form of a univariate polynomial also exists, in which we treat the value of the n input bits as a single value in $GF(2^n)$. [52, 53].

We have decided to focus on the truth tables, with the positions of a 1 and a 0 being swapped as the move function. Not only does this move function preserve balancedness, but several smoothnesses in the search landscape exist for the truth table representation, as we shall demonstrate below:

Lemma 4.3.1. *If one element of the truth table of a Boolean function f with more than one input bit changes value, the algebraic immunity of f changes by at most 1.*

Proof. Let x_α be the input value for which the output value flips. Let f be the original function, f' the function after the truth table is altered that differs from f only in the value of $f(x_\alpha)$. Let g be an annihilator of either f or $(f \oplus 1)$ of degree $AI(f)$.

$f'(x) = f(x) \oplus \delta(x_\alpha)$, where $\delta(x_\alpha)$ is the sum of all supermonoms of x_α. (supermonoms being x_α and all multiples thereof, i.e. any monoms containing all the "on" variables of x_α.)

That is, $\delta(x_\alpha) = x_\alpha(1 \oplus x_b \oplus x_c \oplus x_b x_c \oplus \ldots) = x_\alpha(1 \oplus x_b)(1 \oplus x_c)\ldots$ where x_b, x_c, etc are input bits not appearing in the monom x_α. Let us refer to these as "not-in-common inbits", and the others as "in-common inbits". For example, $\delta(10001) = x_1(1 \oplus x_2)(1 \oplus x_3)(1 \oplus x_4)x_5$, where x_1 and x_5 are the in-common inbits, and x_2, x_3, x_4 are the not-in-common inbits.

$\delta(x_\alpha) \cdot$(one of the not-in-common inbits) $= 0$. (Note that if x_α is the maximum-weight-all-ones input, no not-in-common inbits exist). Furthermore, $\delta(x_\alpha) \cdot (1 \oplus$any in-common inbit$) = 0$.

If $x_b g = 0$ for all not-in-common x_b, g must be a multiple of $(1 \oplus x_b)(1 \oplus x_c)\ldots$, with algebraic degree $\geq (n - HW(x_\alpha))$.

If $(1 \oplus x_i)g = 0$ for all in-common x_i, g must be a multiple of $(x_i \cdot x_j \cdot \ldots) = x_\alpha$, with algebraic degree $\geq HW(x_\alpha)$.

If $x_b g = 0$ for all not-in-common x_b and $(1 \oplus x_i)g = 0$ for all in-common x_i, g must be $x_\alpha(1 \oplus x_b \oplus x_c \oplus x_b x_c \oplus \ldots)$ with algebraic degree n. Since g has algebraic degree $AI(f)$, which is bounded above by $\lceil \frac{n}{2} \rceil$, this is only possible if $n = 1$. So there exists at least one x_b or $(1 \oplus x_i)$ such that the product of it and g is nonzero, and such that the product of it and $\delta(x_\alpha)$ is zero. Call it z.

(In fact, since g must have algebraic degree $\leq \lceil \frac{n}{2} \rceil$, there exist at least $\lfloor \frac{n}{2} \rfloor$ such candidates for z; however we only need one of them to complete the proof.)

Either g is an annihilator of f, or an annihilator of $(1 \oplus f)$.

If the former: $fg = 0$. Then $zgf' = zg(f \oplus \delta) = zgf \oplus zg\delta$. $gf = 0$, so this $= zg\delta = z\delta g = 0$. Hence zg annihilates f', and $AI(f') \leq deg(zg) \leq AI(f) + 1$.

If the latter: $(1 \oplus f)g = 0$. $zg(1 \oplus f') = zg(1 \oplus f \oplus \delta) = zg(1 \oplus f) \oplus zg\delta = 0z \oplus z\delta g = 0$. Hence zg annihilates $(1 \oplus f')$, and $AI(f') \leq deg(zg) \leq AI(f) + 1$.

We have shown that $AI(f') \leq AI(f) + 1$. It is trivial to swap f' and f and repeat the above procedure to show that $AI(f) \leq AI(f') + 1$. Hence $|AI(f) - AI(f')| \leq 1$. □

Lemma 4.3.2. *If one of the 0s in the truth table of a Boolean function f on more than one input bit changes to a 1, and if one of the 1s in said truth table simultaneously changes to a 0, the algebraic immunity of the resultant Boolean function f' differs from $AI(f)$ by at most 1.*

Proof. Since this represents two changes to the truth table of f, we know from the above result that $|AI(f) - AI(f')| \leq 2$. Now, let the first change be the one turning a 1 into a 0 in the truth table, and let the Boolean function resulting from this change be denoted f_2. Clearly any annihilators of f are annihilators of f_2, so $AI(f_2) \leq AI(f)$.

The second change, a 0 to a 1, changes f_2 into f'. From result 10 above, we know that $AI(f') \leq (AI(f_2) + 1) \leq (AI(f) + 1)$. By similar reasoning, we can show that $AI(f) \leq (AI(f') + 1)$. Hence $|AI(f) - AI(f')| \leq 1$. □

Lemma 4.3.3. *Let $DP(f)$ be the minimum value of $d_g + d_h$ such that $f \in B_n$ $(n > 1)$ satisfies a $(d_g, d_h)-$relation. Let f' be a Boolean function differing from f in precisely one truth table position, corresponding to input value x_α.*

Then $|DP(f') - DP(f)| \leq 2$.

Proof. As noted in Lemma 4.3.1 above, $f' = f \oplus \delta(x_\alpha)$, where $\delta(x_\alpha)$, for all input bits $x_b, x_c, \ldots$ that are not submonoms of x_α, is equal to $x_\alpha(1 \oplus x_b)(1 \oplus x_c)\ldots$

Let g with degree d_g and h with degree d_h be two functions such that a $(d_g, d_h)-$relation exists for f. For a valid $(d_g, d_h)-$relation, since $d_h \geq AI(f)$, $d_g \leq \lfloor \frac{n}{2} \rfloor$.

If $x_b g = 0$ for any input bit x_b that is not a submonom of x_α, g must be a multiple of $(1 \oplus x_b)$.

If $(1 \oplus x_i)g = 0$ for any input bit x_i that is a submonom of x_α, g must be a multiple of x_i.

It follows that there must exist at least $\lceil \frac{n}{2} \rceil$ polynomials $p = x_b$ or $(1 \oplus x_i)$ of the form described above such that pg is a nonzero function, otherwise g would have algebraic degree higher than $\lfloor \frac{n}{2} \rfloor$. Let us choose one, and denote it z.

$z \cdot \delta(x_\alpha)$ must equal zero, since if z is one of the x_b, we have $z \cdot \delta = x_\alpha x_b(1 \oplus x_b)\ldots = x_\alpha \cdot 0 \cdot \ldots = 0$, and if z is one of the $(1 \oplus x_i)$, $z \cdot x_\alpha = (1 \oplus x_i)x_i x_j \ldots = 0$ and hence $z \cdot \delta = 0 \cdot (1 \oplus x_b)(1 \oplus x_c)\ldots = 0$.

Now, $zgf' = zg(f \oplus \delta) = zgf \oplus zg\delta = zh \oplus (gz\delta = 0) = zh$. $deg(zg) \leq deg(g) + 1 = (d_g + 1)$, and $deg(zh) \leq deg(h) + 1 = (d_h + 1)$. We see that $DP(f')$ cannot exceed $(DP(f)+2)$ since $(zg)f' = zh$ with $deg(zg)+deg(zh) \leq (d_g+1)+(d_h+1) = (d_g+d_h+2)$.

We can similarly show that $DP(f) \leq (DP(f')+2)$, giving us the result that $|DP(f')-DP(f)| \leq 2$. □

Corollary 4.3.4. *Let $DP(f)$ be the minimum value of $d_g + d_h$ such that $f \in B_n$ $(n > 1)$ satisfies a $(d_g, d_h)-$relation. Let f' be a Boolean function differing from f in precisely two truth table positions. Then $|DP(f') - DP(f)| \leq 4$.*

Lemma 4.3.5. *Let f' be a Boolean function differing from f in precisely one truth table position. Then all values in the Walsh-Hadamard spectrum of f' differ from their corresponding values in the spectrum of f by ± 2.*

Proof. Consider that, as stated earlier, entry ω in the spectrum is equal to:

$$\hat{F}(\omega) = \sum_{i=0}^{2^n-1} (-1)^{f(i)} \cdot (-1)^{\omega \cdot i}$$

Since only one value of $f(i)$ changes, only one value of $(-1)^{f(i)} \cdot (-1)^{\omega \cdot i}$ changes, from either $(-1)\cdot(-1)^{\omega \cdot i}$ to $1\cdot(-1)^{\omega \cdot i}$, or vice versa. In any case, the magnitude of the change is $2\cdot(-1)^{\omega \cdot i}$, i.e. 2. □

Corollary 4.3.6. *Let f' be a Boolean function obtained by swapping two differing values in f's truth table. Then all values in the Walsh-Hadamard spectrum of f' differ from their corresponding values in the spectrum of f by $+4, 0$, or -4.*

Since, as stated earlier, all Walsh-Hadamard spectrum entries for a balanced function are multiples of 4, we have:

Corollary 4.3.7. *Let f' be a balanced Boolean function obtained by swapping two differing values in f's truth table. Let $MW(f)$ denote the maximal absolute value in the Walsh-Hadamard spectrum of f; that is $MW(f) = \max_\omega |\hat{F}(\omega)|$. Then $MW(f') = MW(f)$ or $MW(f) \pm 4$. In any case, the difference is at most 4. Since nonlinearity is defined as $2^{n-1} - \frac{\max_\omega |\hat{F}(\omega)|}{2}$, we see that the nonlinearities of f and f' differ by at most 2.*

Early experiments on evolving truth tables with 8 or 9 input bits showed that the optimal values for AI and FAR would always be found within two outer loops, even with only 100 inner loops. For this reason, we felt confident in focusing solely on truth tables, and in adding nonlinearity to the cost function, thus covering all the relevant criteria for a filter function [55].

4.3.2 Choosing a cost function

When evolving Boolean functions with high nonlinearity, Clark et al. experimented with cost functions of this form [80] for various values of R and X:

$$cost(f) = \sum_{\omega=0}^{2^n-1} ||\hat{F}_f(\omega)| - X|^R$$

To be more precise, the value $R = 3.0$ was preferred, with 2.0 and 2.5 also experimented with. In devising the part of the cost function that would deal with nonlinearity, however, we opted to utilise $R = 4.0$ (and to divide this part of the cost function by a scalar factor dependent on n), for various reasons:

1. According to Parseval's Theorem, the sum of squares of the entries in a valid Walsh spectrum is constant. It therefore seemed unlikely that exponent 2 would be of much help. Furthermore, we had observed that high-quality solutions tended to have higher costs as defined by the pair $(X = 0, R = 1)$; and although attempts

to base a cost function on this observation proved ineffective, this was nonetheless evidence that R would have to exceed 2.

2. Applying a matrix transformation to the difference distribution table (DDT) of a vectorial Boolean function yields a table containing the autocorrelation spectra of all linear combinations of the co-ordinate functions, and applying a further matrix transformation to this yields a table containing the squared entries of the Walsh-Hadamard spectra for these functions [141]. Our research into evolving substitution boxes (see Chapter 5) had utilised the sum of squares of DDT entries after ($R = 2.0, X = 0$) for this table turned out to be especially efficient and high-performing, and this suggested that the sum of the squares of the squares of the Walsh entries might be analogous with the sum of the squares of the DDT entries for a vectorial Boolean function in some way.

3. Consistent with the preceding point, dividing the variance of the entries in the "squared Walsh spectra" table by a particular value exponential in n yielded the variance of the DDT; and we had been able to prove that the cost as defined by the DDT variance changed by the same amount as the ($R = 2.0, X = 0$) DDT cost function whenever a move was made.

4. During initial experimentation, dividing the sum of fourth powers by 2^{n+5} to define a cost was observed to create a situation where each move changed the cost by 3.0 or some integer multiple thereof, raising confidence in the uniform smoothness of the search landscape.

5. Furthermore, when combined with algebraic and fast algebraic qualities, this cost function obtained Boolean functions with comparable algebraic characteristics and superior nonlinearity to a cost function in which ($2^{n-1} - NL$) - (the number of occurrences of the maximal absolute value in the Walsh spectrum) was used as the nonlinearity component.

The overall cost function, therefore, derived an initial cost using the Walsh spectrum in this fashion, and then subtracted $2 * AI(f) + FAR(f)$ from it to obtain the overall cost. This meant that a one point improvement in the nonlinearity portion of the cost function would subtract 3 from the cost, compared to 1 or 2 for the others. We felt that this was justified to reflect the difficulty of obtaining functions with optimal nonlinearity

through simulated annealing compared to functions with optimal algebraic characteristics. In experiments, it was observed that this would allow the cost function to move through candidates with suboptimal algebraic characteristics that might otherwise block off promising search avenues. The additional weight given to AI compared to FAR simply reflected its more restricted range of values.

As stated above, we used a different cost function for hill-climbing. This, again, subtracted $2 * AI(f) + FAR(f)$ from the overall cost, but had a simpler nonlinear component of $(2^{n-1} - NL) - 2/freq(max_f(|\hat{F}(\omega)|))$. That is, we divided 2 by the frequency with which the maximal absolute value in the Walsh spectrum occurred, and subtracted this from $(2^{n-1} - NL)$. On this occasion, however, we reduced the weighting given to the nonlinearity - slightly suboptimal nonlinearity was acceptable, anything less than optimal AI and FAR in the final product was not.

We used 500,000 inner loops for problems of size 9 or higher, and 20,000 for size 8 or less. We used 100 outer loops and 50 trials per problem size, cooling factor 0.97, and initial acceptance rate 0.5. Algebraic immunity was calculated using Carlet, Meier and Pasalic's Algorithm 2 [57], and fast algebraic resistance using the algorithm of Braeken, Lano and Preneel [43].

The next cost function

For up to 11 input bits, this was acceptably efficient. The following table compares our results to the previously-known best in the literature [55, 56, 58, 148]:

n	Previous best (NL, AI, FAR)	Best (NL, AI, FAR) achieved by annealing
6	(24, 3, 5)	(26, 3, 5)
7	(54, 4, 6)	(56, 4, 6)
8	(114, 4, 7)	(116, 4, 7)
9	(236, 5, 8)	(238, 5, 8)
10	(484, 5, 9)	(486, 5, 9)
11	(980, 6, 10)	(986, 6, 10)

Table 4.1: Comparisons of annealed and previously-known Boolean functions for $n \leq 11$.

However, both in memory and time, the cost of calculating algebraic immunity and fast

algebraic resistance is exponential. Despite the optimisations we were able to make by taking into account the lemmas above, both complexities were still exponential, and for 12 input bits the algorithm remained stuck in its first hill-climb for several days without returning a result.

Since most of the results that had been achieved still had optimal algebraic characteristics, and since the speed with which these were achieved suggested that functions with optimal (AI, FAR) were plentiful, we decided to run a new set of experiments in which we would remove all parts of the cost functions that did not focus on nonlinearity. We would evaluate (AI, FAR) at the end of the algorithm, and hope that at least some of the annealed functions were optimal in terms of these criteria.

The parameters remained unchanged up to $n = 15$. For $n = 16$, the increased complexity meant that we reduced the number of inner loops to 200,000; however we later raised this to 400,000 (and later 1,000,000, after discovering the substantial gulf between constructed and annealed results at this size.) We did not go as far as $n = 17$; and note that to do so would require at least $4GB$ of memory for the fast algebraic resistance calculations and the precomputed tables used in the nonlinearity sections of the cost function; this quantity increases approximately fourfold when n is increased by 1.

We also reran the experiments for $n = 9$, $n = 10$ and $n = 11$ using this approach, hoping either to improve on our best results or to increase the number of distinct affine equivalence classes possessing the same set of optimal criteria. For $n = 9$, 8% of functions achieved nonlinearity 240, but all of these had only $FAR = 7$. 32% of the functions for $n = 10$ achieved nonlinearity 488, again at the cost of a slightly suboptimal $FAR = 8$. The new experiment for $n = 11$, after hill-climbing, found functions with nonlinearity 988 on every run, but none of these possessed the necessary $FAR > 9$. What was more, as well as $FAR(f) = (n - 2)$, these functions also had suboptimal algebraic degree $(n - 2)$.

Comparing this to the results for higher sizes; for $n = 12$ 58% of the hill-climbed functions had nonlinearity 1996, but all of these had suboptimal degree and FAR of 10. For $n = 13$ 60% of the hill-climbed functions had nonlinearity 4020, but all of these had FAR 11 and degree 11.

n	Best (NL, AI, FAR) achieved with nonlinearity-only cost function.
12	(1994, 6, 11)
13	(4018, 7, 12)
14	(8082, 7, 13)
15	(16222, 8, 14)
16	(32536, 8, 15)

Table 4.2: Annealed Boolean functions for $12 \leq n \leq 16$ before incorporation of algebraic degree into the cost function.

4.3.3 Adding algebraic degree to the cost function.

Since all the functions we had found with nonlinearity in excess of those in Table 4.2 had suboptimal algebraic degree, we altered the hill-climb cost function to heavily penalise algebraic degree $< (n-1)$, and reran the previous experiments with increased numbers of inner loops (going as far as 16,000,000 for $n = 14$).

The results of this were mixed. For $n \leq 13$, the higher values for nonlinearity observed previously simply did not occur. For $n = 14$, two Boolean functions with nonlinearity 8084 and the desired $(AI,\ FAR)$ value were obtained; all the other functions at this size had nonlinearity 8082. For $n = 16$ (with up to 3,000,000 inner loops) one function with nonlinearity 32540 has been found. No functions with higher nonlinearity at this size have yet been obtained through annealing; however all functions with this or lower nonlinearity have so far possessed optimal $(AI,\ FAR)$, suggesting that experiments over a longer time period with more inner loops may obtain higher nonlinearities still.

For $n = 15$ (with up to 2,000,000 inner loops), however, most annealed functions had only suboptimal AI of 7, despite their optimal degree and FAR. One function with (NL 16226, AI 8, FAR 14) has nevertheless been found, but the reduced AI of most of the results suggests that very few Boolean functions with high nonlinearity possess optimal algebraic degree, algebraic immunity and fast algebraic resistance at this size, and that increasing the computational resources devoted to this problem with the current cost function might primarily have the effect of reducing the number of functions with $AI = 8$. It should be noted that the evaluation of a Boolean function's algebraic immunity is much slower than the evaluation of its algebraic degree, and hence reintroducing this into the cost function would significantly increase the time required to anneal a single Boolean function, or force a reduction in the number of inner loops (and hence the achievable nonlinearity). This may even result in functions with optimal algebraic degree $(n-1)$ but

$FAR \leq (n-2)$.

n	Previous best (NL, AI, FAR)	Best (NL, AI, FAR) achieved with annealing.
12	(1988, 6, 11)	(1994, 6, 11)
13	(3988, 7, 12)	(4018, 7, 12)
14	(8072, 7, 13)	(8084, 7, 13)
15	(16212, 8, 14)	(16226, 8, 14)
16	(32556, 8, 15)	(32540, 8, 15)

Table 4.3: Comparisons of the best existing Boolean functions with the final annealing results for $12 \leq n \leq 16$.

4.3.4 Equivalence classes

The histograms of the values in the Walsh spectra of the evolved functions differed, even for functions with the same (NL, AI, FAR). Since these frequency histograms are affine invariant, it was clear that several different affine equivalence classes of functions existed with these properties.

(n, NL, AI, FAR)	Number of distinct equivalence classes identified
(6, 26, 3, 5)	2
(7, 56, 4, 6)	2
(8, 116, 4, 7)	10
(9, 238, 5, 8)	62
(10, 486, 5, 9)	53
(11, 986, 6, 10)	11
(12, 1994, 6, 11)	22
(13, 4018, 7, 12)	7
(14, 8084, 7, 13)	2
(15, 16226, 8, 14)	1
(16, 32540, 8, 15)	1

Table 4.4: Number of non-equivalent functions so far with the best (NL, AI, FAR) obtained through annealing.

4.4 Summary of achievements, and avenues for future research

In this chapter, we have established via theoretical analysis that the search landscape defined by the use of truth table flips as a move function is extremely promising with respect to the search for Boolean functions resistant to algebraic attacks, and indeed (building on existing results in this area) with respect to Boolean functions resistant to stream cipher cryptanalysis in general. We have demonstrated the existence of smooth search landscapes for algebraic immunity and fast algebraic resistance, and exploited these alongside the already known smooth landscape for nonlinearity, in a local-optimisation based metaheuristic which has found Boolean functions with superior properties to the best theoretical constructions for their corresponding values of n.

Truth tables for some of the evolved Boolean functions are presented in a later appendix, and any researchers wishing to investigate the full set of evolved truth tables are invited to email the authors.

It would be interesting to see if such a search landscape is also defined for properties such as transparency order which are relevant to side-channel attacks, or indeed for any other properties of Boolean functions that are cryptographically relevant. Or, for that matter, relevant in areas of computer science other than cryptology.

The key issue with the new functions is one of implementation. The Carlet-Feng functions can be implemented using the Pohlig-Hellman algorithm [134] for up to 20 bits (and possibly more) without needing the truth-table to be stored in memory; and for purposes of efficiency, some fast means to calculate one of the new functions without needing to store a large lookup table in memory or requiring a circuit with an overly large number of gates is required for them to be of practical use. Algebraic immunity is not invariant in the case of affine transformations on the outputs, but is invariant under transforms on the function inputs, and all other relevant properties are affine invariant [43]. Hence, a potentially profitable avenue might be to apply various affine transformations to the function inputs and to experiment with the results to find out if any of them are of the types described by Carlet et al. [55, 56, 58]. Alternatively, the univariate representations of the affine equivalence subclasses thus defined could be examined for functions with suitably sparse univariate forms.

In this chapter, Boolean functions satisfying all the required properties for use as

nonlinear filter functions, and with nonlinearity higher than that achieved by existing constructions, have been shown to exist. Now the question is whether any of them can be shown to be part of an infinite class of Boolean functions with these properties (and, ideally, some more efficient means of implementation). The exponential complexity of the algebraic immunity and fast algebraic resistance algorithms renders the use of the current annealing approach to find such functions for higher values of n increasingly impractical.

We note that faster and more memory-efficient algorithms for the calculation of algebraic immunity and fast algebraic resistance could increase the values of n to which this algorithm can be applied, as well as the algorithm's performance for the values of n we have so far been able to apply it to.

Although addressing implementation issues would go beyond the scope of the work in this chapter, we have successfully demonstrated the existence of functions with hitherto-unobtained cryptographic properties, and the power of metaheuristics to find these. Our work has shown that metaheuristic search has very significant potential in the construction of Boolean functions with excellent cryptographic properties.

Chapter 5

Using evolutionary computation to create vectorial Boolean functions with low differential uniformity and high nonlinearity.

The previous chapter dealt with Boolean functions mapping several input bits to only one output bit. In this chapter we attempt to generalise to Boolean functions with multiple-bit output values, and to address the different criteria such functions must satisfy. There has been previous research in the use of metaheuristics in this area, with limited success, but we believe there is a need to focus on different criteria to those of the previous researchers. We also utilise three different types of metaheuristic - simulated annealing, memetic algorithms, and ant algorithms - and three different types of algorithm from the third category. Memetic algorithms have parameters which are themselves choices of algorithm, and we vary these as well. Initial experiments focus on finding suitable cost functions, then on finding the best parameter choices for the various algorithms. Finally, we compare the performance of the algorithms.

5.1 Introduction

The two most important criteria for vectorial Boolean functions used in block ciphers are differential uniformity and nonlinearity. Previous research into evolving such functions us-

ing metaheuristics has focused only on nonlinearity and a different criterion, autocorrelation. In this chapter, we describe the results of experiments in using simulated annealing, memetic algorithms, and ant colony optimisation to create vectorial Boolean functions with low differential uniformity.

5.1.1 Technical background

Definition 5.1.1. For some n, m, an $n \times m$ S-box is a mapping from $GF(2^n)$ to $GF(2^m)$. If every value $\in GF(2^m)$ is mapped to by an equal number of distinct input values, we say that the S-box is *balanced*.

S-boxes are typically the sole nonlinear components of ciphers, applied repeatedly so that the equations describing each ciphertext bit will be complicated equations of high algebraic degree in all the plaintext and key bits involving large numbers of monomials. Although the original Data Encryption Standard [191] used S-boxes mapping a six-bit output to a four-bit input, most modern cipher designs use only bijective $n \times n$ S-boxes. In particular, we note that the current Advanced Encryption Standard (AES) [192], and most block ciphers designed for environments where the AES is too resource-intensive (e.g. PRESENT [40], PRINTCIPHER [162]), use only bijective $n \times n$ S-boxes. For this reason, we will focus primarily on these S-boxes.

Even if it is not bijective, the S-box is usually required to be balanced. Otherwise, the two most important criteria for a cryptographically secure S-box are low *differential uniformity* and high *nonlinearity*. There are other relevant criteria as well - as an example, it has been suggested that if large numbers of low-degree implicit equations with the S-box's input and output bits as variables hold, this may lead to the cipher being broken [110, 184].

Definition 5.1.2. Let S be an S-box. Let us construct a table with 2^n rows and 2^m columns by defining the entry in row i, column j to be the number of inputs x such that $S(x) \oplus S(x \oplus i) = j$. ($\oplus$ will signify exclusive-or throughout this chapter.) This table is known as the *difference distribution table*, or DDT.

Note that all entries in this table must be even, for if $S(x) \oplus S(x \oplus i) = j$, then let $y = (x \oplus i)$, and we see that $S(y) \oplus S(y \oplus i) = j$. Furthermore, it is clear that all entries in the row of a DDT must sum to 2^n, and hence that the sum of all entries in all rows must be equal to 2^{m+n}.

Definition 5.1.3. The *differential uniformity* of an S-box S, sometimes denoted $DU(S)$ (or just DU when it is clear from the context that S is meant), is the largest value present in its difference distribution table (barring the entry for $i = j = 0$).

Where $DU(S) = k$, we refer to S as being *differentially k-uniform.*

Definition 5.1.4. An S-box with differential uniformity 2 is referred to as being *almost perfect nonlinear* (or "APN"). In theory, perfect nonlinearity would correspond to DU 1, however since all entries in the DDT must be even, this is not achievable in practice. The term "almost perfect nonlinear" was introduced by Vaudenay et al. [61] in 1994.

Definition 5.1.5. The number of times that the value D equal to $DU(S)$ appears in the difference distribution table of S is referred to as its *differential frequency* (DF for short).

Definition 5.1.6. Let S be an S-box. Let us construct another table with 2^n rows and 2^m columns by defining the entry in row i, column j as follows: where the parity of i AND x is equal to the parity of j AND $S(x)$ for k values of x, the corresponding entry in the table is the value $k - 2^{n-1}$.

This table is known as the *linear approximation table*, or LAT. The proof is a little too long to reproduce here, but as long as the S-box is balanced, this table also has the property that all its entries are even.

Definition 5.1.7. The *nonlinearity* of an S-box, $NL(S)$ (or just NL when the box involved is clear from the context.) is equal to $2^{n-1} - max_{i,j}|LAT_{ij}|$ - that is, 2^{n-1} minus the maximum absolute value in the LAT (excluding the entry for $i = j = 0$.)

Clearly this generalises the definition of nonlinearity from the previous chapter.

Definition 5.1.8. The number of times that the value $2^{n-1} - NL(S)$ appears in the linear approximation table of S is referred to as the *nonlinear frequency* of S (NF for short).

Differential uniformity and nonlinearity are the most important properties in a cipher's S-boxes, being crucial to its resistance against differential cryptanalysis [32, 34] and linear cryptanalysis [179, 180] respectively, as well as the many variants on and hybrids of these techniques.

As stated earlier, previous research also focused on a property known as *autocorrelation.* While this has some relevance to the cipher's resistance against the hybrid technique *differential-linear cryptanalysis* [133, 29], differential uniformity and nonlinearity are far more important even in this context. We did not therefore focus on autocorrelation in

our experiments, but for the sake of completeness when discussing previous research we provide the relevant definitions here:

Definition 5.1.9. Where S is an S-box, let us construct a third table with 2^n rows and 2^m columns. Let $S_j(x)$ be the Boolean function with one output bit defined by xoring the output bits of S corresponding to the 1s in the bitmask j. Let the entry in row i, column j be equal to the number of inputs for which $S_j(x) = S_j(x \oplus i)$, minus the number of inputs for which this was not the case.

This table is the *autocorrelation table*, or ACT.

Definition 5.1.10. The *autocorrelation* of an S-box, $AC(S)$ (or just AC when it is clear from the context that S is the box involved) is the maximum absolute value in its autocorrelation table, barring the trivial entry for $i = j = 0$.

Definition 5.1.11. The number of times that the value $AC(S)$ appears in the autocorrelation table of S is referred to as the *autocorrelation frequency* of S (AF for short).

The three tables are in fact related. Using matrix transformations, the ACT can be derived from the DDT, and a table C such that $C_{i,j} = (2 \cdot LAT_{i,j})^2$ can be derived from either the LAT or the ACT [141]. Since this will prove relevant later on, we explain it in more detail here.)

Definition 5.1.12. The Walsh-Hadamard matrix is also known as the "Sylvester-Hadamard matrix". It is defined recursively thus:

$$H_0 = \begin{bmatrix} 1 \end{bmatrix}$$

$$H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -1 \times H_{n-1} \end{bmatrix}$$

So, for example,

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Theorem 5.1.13. *For an $n \times m$ S-box S, let D_S denote the DDT of S. Treat it as a matrix for the purposes of multiplication; then $H_n D_S H_m$ is the aforementioned table C such that $C_{i,j} = (2 \cdot LAT_{i,j})^2$.*

For a proof, the reader is referred to the paper in which this theorem originally appeared [141].

5.1.2 The use of metaheuristics in this field.

Although there was some earlier work using simpler techniques such as hill-climbing, and although metaheuristics had been applied to the related problem of evolving single-output-bit Boolean functions, the first major attempt to construct S-boxes by utilising metaheuristics was carried out in 1999 by Millan et al. [48]. They attempted to maximise nonlinearity and minimise autocorrelation - nonlinearity itself being used as the fitness function for the NL experiments, and some value derived from the autocorrelation as the fitness function for the experiments in that area. (It is not clear from the paper what this fitness function was; most likely either $1/AC$ or the subtraction of AC from some fixed value.) The metaheuristic used was a genetic algorithm variant without mutation and with an unconventional crossover function that the authors claimed introduced enough randomness to eliminate the need for mutation.

Encouraged by the success of their previous research into the use of simulated annealing to evolve Boolean functions with 1-bit output length and low autocorrelation/high nonlinearity [87, 84] [80], Clark, Jacob and Stepney employed the same techniques to try to obtain S-boxes with these qualities [86]. Their approach differed from that which had preceded it in two key respects:

- Instead of simply using the value of nonlinearity/autocorrelation for the cost/fitness function, the whole linear approximation/autocorrelation table would be taken into account, with the cost function focusing on trying to bring every value close to zero.
- After using one cost function for the simulated annealing, they would end the search by hill-climbing, usually with a different cost function - so, for example, after obtaining a relatively flat linear approximation table, they would hill-climb with nonlinearity as the fitness function, based on the hypothesis that the primary cost function would have guided the search into a region of the search space in which a higher number of candidate solutions with above average nonlinearity existed.

This approach improved on the best nonlinearity obtained by Millan et al. for bijections such that $6 \leq n = m \leq 8$, and equalled it for $n = m = 5$. However, the researchers noted that S-boxes with higher nonlinearity were known to exist for $n = 6$ and $n = 8$. They also noted that a slight increase in the value of n resulted in a massive increase to the problem complexity.

S-boxes with equivalent properties and transformations.

Various equivalence notions exist according to which there may be several S-boxes in the search space of bijections over $GF(2^n)$ with identical differential uniformity and nonlinearity; and indeed with the same sets of absolute values in their DDTs and LATs [46]. This has the potential to be particularly problematic for genetic and memetic algorithms, since it implies that many different "genes" may result in the same cryptanalytically relevant properties (see Appendix B for more information on this.) We define these notions here:

Definition 5.1.14. Let S_1, S_2 be two S-boxes.

S_1 and S_2 are *affine-equivalent* iff $S_2 = A{\cdot}S_1{\cdot}B$, where A, B are bijective affine transformations (so $A(x)$ would be the result of applying the transformation $M_A x \oplus V_A$, where M_A was some invertible matrix and V_A a vector.)

Definition 5.1.15. Let S_1, S_2 be two S-boxes.

S_1 and S_2 are *extended affine-equivalent* (EA-equivalent) iff $S_2 = A{\cdot}S_1{\cdot}B \oplus C$, where A, B are bijective affine transformations and C is some, not necessarily bijective, affine transformation acting on the same input x as $A{\cdot}S_1{\cdot}B$.

Definition 5.1.16. Let S_1, S_2 be two S-boxes.

Let $gr(S_1)$ denote the graph of S_1, i.e. the set of all $(x, S_1(x))$ pairs. Let each such pair be viewed as a value in $GF(2)^{2n}$.

S_1 and S_2 are *Carlet-Charpin-Zinoviev equivalent* (CCZ-equivalent for short) iff there exists some affine permutation $L : GF(2)^{2n} \to GF(2)^{2n}$ such that $L(gr(S_1)) = gr(S_2)$.

(so $L(x, S_1(x)) = (L_1(x, S_1(x)), S_2(L_1(x, S_1(x))))$, where $L_1 : GF(2)^{2n} \to GF(2)^n$ maps the input bits of L to the first n output bits.)

Definition 5.1.17. An *affine-invariant* property is a property which, if possessed by an S-box S, is also possessed by all S-boxes affine-equivalent to S. Similarly, an EA-invariant property is a property which, if possessed by an S-box S, is also possessed by all S-boxes

EA-equivalent to S, and CCZ-invariant properties are defined in terms of CCZ-equivalent S-boxes in the same way.

It may be seen from the above definition that any two S-boxes which are affine equivalent are also EA-equivalent. It follows from this that all EA-invariant properties are also affine-invariant. Furthermore, CCZ-equivalence generalises EA-equivalence [46], so all CCZ-invariant properties are also EA-invariant and hence affine-invariant.

Importantly, differential uniformity and nonlinearity are CCZ-invariant properties [46].

We sought to find ways of utilising the equivalence classes to reduce the size of the search space, and to reveal patterns in the truth tables of the evolved mappings which could be exploited by the ant colony and memetic algorithm experiments (as well as some experiments with genetic algorithms that were later superseded by the memetics). The following theorem resulted from this:

Theorem 5.1.18. *Every bijective S-box S is affine-equivalent to at least one bijective S-box S_2 such that S_2 maps all inputs with Hamming weight less than 2 to themselves, 3 to 3 or 5 (we can restrict this to 5 if the S-box has differential uniformity ≤ 4), 5 to some value ≤ 11 (this may be restricted to 6 or 10 if the S-box has differential uniformity 2), and all $2^i + 1 (3 \leq i \leq (n-1))$ to some value $\leq 2^{i+2} - 2i - 1$.*

The proof, which also describes the procedure to construct the equivalent S-box, is included in Appendix B.

5.2 Experiments with simulated annealing

We adopted the approach of Clark et al. of annealing with one cost function, and then hill-climbing with another. We also based our annealing cost functions on theirs [86], in that for whichever table T was relevant to the criterion of primary interest the cost function took the form:

$$\sum_{i=0}^{2^n-1} \sum_{j=0}^{2^m-1} ||T_{i,j}| - X|^r$$

This family of cost functions aimed to "flatten" the contents of the table T as much as possible, achieving a relatively uniform table in which few entries deviated significantly from the value X. For a difference distribution table in particular, X equal to 0 or 1 would

be an intuitive choice, since for the best possible APN S-boxes all values were 0 or 2 with mean 1, and since low values in this table were desirable. In focusing on all the values in the table, instead of the single most extreme value, the search was better equipped to optimise table values that were not the extremal values at that particular point in time.

As stated earlier, we focused on differential uniformity and nonlinearity, meaning that T would be either the DDT or the LAT. We experimented with various values of X and r for both the LAT and DDT, and tried multiplying and adding the cost function output values for different tables to achieve a multiobjective optimisation.

Our hill-climbing cost functions were not based solely on whichever of the differential uniformity and nonlinearity was being targeted. Since each of these was defined in terms of some extremal value in its corresponding table, we adapted the cost function to subtract $k/$(the number of times this value appeared) - where k was the largest value for that table that could be guaranteed to divide all entries in it. (k was equal to 2 in both cases). This gave us as cost functions:

- $DU - 2/DF$ and
- $max_{i,j}|LAT_{ij}| - 2/NF$

By minimising the respective frequencies, we hoped to minimise the number of ways in which the cryptanalyst could exploit the extreme table values - in particular, we hoped to improve the S-box's resistance against linear cryptanalysis with multiple approximations [38, 70, 66]. We also hoped that this would guide the search towards lower values for the DU and $max_{i,j}|LAT_{ij}|$. Part of this was based on the fact that using only DU or $max_{i,j}|LAT_{ij}|$ to define the cost function would have made it harder for the hill-climbing stage to find improved candidate solutions if there were none with lower cost within the 1-move neighbourhood of the current candidate - in fact, whether it could do so at all would be dependent on the precise hill-climbing method implemented. Another motivation was the fact that the 1-move neighbourhoods of the known APN S-boxes contained differentially-4-uniform S-boxes with extremely low differential frequencies.

5.2.1 Refining the annealing cost functions for differential uniformity

During experiments with the following cost functions:

$$\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}||DDT_{i,j}|-X|^2$$

we observed that where the random number generator (the boost::mt19937 Mersenne Twister from the C++ Boost libraries [1]) had been seeded with the same seed, experiments for different values of X were producing the same results.

To work out why this was, we first note that there are no negative values in the DDT, so the above equation is equivalent to $\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}|DDT_{i,j}-X|^2$. Since, for any d, $d^2=|d|^2$, we have:

$$\begin{aligned}
&\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}||DDT_{i,j}|-X|^2\\
&=\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}|DDT_{i,j}-X|^2\\
&=\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}(DDT_{i,j}-X)^2\\
&=\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}(DDT_{i,j}^2-2{\cdot}X{\cdot}DDT_{i,j}+X^2)\\
&=\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}DDT_{i,j}^2-2X\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}DDT_{i,j}+\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}X^2
\end{aligned}$$

Note that $\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}X^2$ is a constant value, independent of any values present in the DDT. Moreover, $\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}DDT_{i,j}$ was shown in Definition 5.1.2 to be equal to 2^{m+n}, and hence $2X\sum\sum DDT_{i,j}$ is equal to $2^{m+n+1}X$; another constant.

We see that the only part of the above cost function which can change at all when a move is made is $\sum_{i=0}^{2^n-1}\sum_{j=0}^{2^m-1}DDT_{i,j}^2$, and this is not dependent on the value of X.

Early experiments to compare the effects of the various DDT cost functions used:

- $n=6$,
- Cooling factor 0.99 (we would later reduce this to 0.97),
- $MAX\_INNER\_LOOPS=10000$,
- $MAX\_OUTER\_LOOPS=500$,

- $MAX\_FROZEN\_OUTER\_LOOPS = 200$.

These parameters had already been observed to find S-boxes with optimal differential uniformity and nonlinearity in some cases for $n = 5$; however we had not managed to find S-boxes with the optimal properties for $n = 6$, and we felt that more information might be obtained from the differential frequencies of those S-boxes which achieved near-optimal differential uniformity of 4. In all cases, the hill-climb cost function was $DU - 2/DF$.

	X				
	-2	-1	0	1	2
$r = 2$	(59, 224.44)	(59, 224.44)	(59, 224.44)	(59, 224.44)	(59, 224.44)
$r = 3$	(65, 224.68)	(71, 223.1)	(58, 225.72)	(59, 222.71)	(71, 219,97)
$r = 4$	(71, 223.69)	(68, 223.5)	(67, 223.75)	(76, 229.3)	(75, 232.54)
$DU - 2/DF$ to anneal	(0, N/A)				
$r = 2,\ X = 0$ optimised	(78, 217.12)				

Table 5.1: (Percentage of DU 4, average DF for DU 4) for $n = 6$. No boxes with DU 2 were found.

The best DU we found in any of our experiments for $n = 6$ was 4, even though S-boxes of this size with DU 2 are known to exist [45]. Exponent 2 resulted in fewer DU 4 S-boxes being found than the other exponents; however the average DF of these was not significantly different. However, using exponent 2 and $X = 0$ allowed us to code a highly optimised version of the "lookahead" function that evaluated the change in cost when a move was being considered (the most generic approach to the lookahead would be to make the move, update relevant data, call the cost function, then undo the preceding) - and the result of this was that the search concluded 11.81 times as fast as the fastest of the other cost functions. Increasing MAX_INNER_LOOPS proportionately, we obtained 78% DU 4 with average DF 217.12. We therefore accepted $r = 2,\ X = 0$ as the best cost function to use in large scale attempts to obtain low differential uniformity through simulated annealing.

5.2.2 Relating nonlinearity and differential cost functions.

The formula $\sum_{i=0}^{2^n-1} \sum_{j=0}^{2^m-1} ||T_{i,j}| - X|^r$ bears a strong resemblance to the formula for the sample variance of the entries of T. Indeed, for the DDT, let $r = 2$, and let $X = 1$, since this is the sample mean of the entries in the DDT of a bijective S-box, then apart from

the division by 2^{n+m}, the formulae are identical. Prior to our discovering that the value of X did not affect the behaviour of the search algorithm for the DDT with exponent 2, this led to our experimenting with the variance of the DDT as a cost function. During these experiments, the variance of the absolute LAT values, as well as the variance of the squared LAT values (due to Theorem 5.1.13) was also recorded, and we noticed that in every case the latter was equal to $2^{2n-4}\times$ the DDT variance. However, the ratio between the two variances was not a power of 2, or indeed an integer, for non-bijective S-boxes. Investigating further, the same relationship was observed to hold between the sum of squares of the DDT and the sum of fourth powers of the LAT for a bijective S-box. Again, however, this was not the case for a non-bijective S-box.

We therefore make the following conjecture: For bijective S-boxes, the value of DDT cost function $r = 2, X = 0$ is always a fixed multiple of the value of LAT cost function $r = 4, X = 0$ (the precise multiple being determined by the value of n.) Although the temperatures required to achieve the desired initial acceptance rate may differ for these two cost functions; neither should be any more or less effective than the other in simulated annealing - they are in practice equivalent.

In the experiments shown for the 6×6 problem size in Table 5.2, $(r = 4, X = 0)$ is not the best-performing cost function. Nevertheless, using the DDT cost function with $(r = 2, X = 0)$, the fastest of the other cost functions was 554 times slower in completing a search than it was. Increasing MAX_INNER_LOOPS accordingly, as may be seen in the second-last row of the table, results in the best average NF for NL 22. More generally, using the DDT cost function for the annealing stage offers the following advantages:

1. It allows us to use a lower exponent than we would otherwise, reducing the extent to which higher exponents would slow down the exponentiation involved.

2. As stated before, the optimised lookahead algorithm for DDT exponent 2 and $X = 0$ can be used.

3. Updating the difference distribution table for a candidate solution when a move is made requires $O(2^n)$ time. Updating the linear approximation table requires $O(2^{n+m})$ time. Similarly, the initial calculation of these tables also differs in complexity by a factor of 2^m.

We therefore accepted this cost function as the best candidate for larger-scale searches both for S-boxes with high nonlinearity and with low differential uniformity. Two sets

	X						
	-6	-4	-2	0	2	4	6
r=2	(55, 59.13)	(72, 61.46)	(67, 59.78)	(N/A, N/A)	(73, 61.53)	(70, 58.69)	(70, 58.31)
r=3	(99, 48.46)	(100, 41.94)	(100, 41.22)	(100, 37.18)	(95, 50.4)	(56, 62.98)	(59, 50.61)
r=4	(100, 40.83)	(100, 41.04)	(100, 40.24)	(100, 34.3)	(95, 49.4)	(58, 64.98)	(59, 61.3)
r=5	(100, 41.35)	(100, 41.78)	(100, 40.8)	(100, 32.7)	(96, 49.7)	(73, 61.11)	(62, 63.42)
r=6	(100, 42.18)	(100, 43.28)	(100, 42.65)	(100, 32.45)	(98, 52.02)	(71, 59.92)	(64, 64.88)
DDT ($r = 2,\ X = 0$) with speed/power tradeoff				(100, 30.39)			
$max_{i,j}\|LAT_{ij}\| - 2/NF$ used to anneal				(100, 59.53)			

Table 5.2: (Percentage of NL 22, average NF for NL 22) for $n = 6$. No NL 24 were found.

of experiments, each consisting of 100 runs, were carried out. These used $\alpha = 0.97$ and, respectively, 3,000,000 and 30,000,000 inner loops per value of temperature.

max. inner loops	% DU 2 after DU/DF hill-climb	% NL 12 after NL/NF hill-climb	% (DU 2, NL 12) after dual hill-climb
3,000,000	11	11	6
30,000,000	45	34	35

Table 5.3: Experiments for $n = 5$. For this size, NL cannot exceed 12 [61].

Inner loops	(% DU 4, avg. DF for DU 4) after DU/DF hill-climb	(% NL 22, avg. NF for NL 22) after NL/NF hill-climb	% (DU 4, NL 22) after dual hill-climb
3,000,000	(80, 204.25)	(100, 29.83)	35 (avg. DF 204, avg. NF 36.09)
30,000,000	(86, 197.65)	(100, 28.06)	36 (avg. DF 199.25, avg. NF 31.78)

Table 5.4: Experiments for $n = 6$. For this size, (DU 2, NL 24) and (DU 4, NL 24) S-boxes are known to exist but were not found. It is not known if NL 26 boxes exist.

For the $n = 5$ problem size, the APN S-boxes found fell into two categories. The boxes with nonlinearity 12 have properties consistent with the Gold exponent S-boxes [65] and their inverses, whereas the APNs with nonlinearity 10 appear to be "inverse-based" S-boxes [65]. All S-boxes with nonlinearity 12 were APN, this being due to the properties of almost bent S-boxes [61]. This means that we have, for this problem size, obtained results matching the theoretical best-possible (and the best S-boxes obtained through mathematical construction), as well as improving on previous applications of simulated annealing to this problem (which only achieved nonlinearity 10 - see Table 5.5.)

The best S-box we achieved for $n = 6$ in terms of differential uniformity had DU 4, DF 159, NL 22, and NF 24, occurring as part of the 30,000,000 inner loop experiment. Unfortunately, since a 6×6 APN S-box has been constructed [45], this means that we have not managed to match the best existing constructions for this size.

Although 6×6 S-boxes with nonlinearity 24 have been constructed mathematically [197] [45], we were unable to obtain any such for any choice of cost function. The best we were able to achieve had nonlinearity 22. The lowest corresponding NF we found was 22 (unfortunately in conjunction with DU 6).

For larger problem sizes, we were unable to compete with the best known existing

results. We did not achieve nonlinearity higher than 48 for $n = 7$, or higher than 104 for $n = 8$ (the best-known values for these sizes are 56 and 112). The best DU possible for $n = 7$ is 2, we achieved only DU 6 with best DF 11. Similarly, the best known DU for $n = 8$ is 4; the best we achieved was 6 with DF 185.

	Highest known NL	Previous best evolved NL [86]	Highest evolved NL (this thesis)
$n = 5$	12	10	12
$n = 6$	24	22	22
$n = 7$	56	48	48
$n = 8$	112	104	104

Table 5.5: Comparison of best-known and evolved nonlinearities for $n = 5$, 6, 7 and 8.

5.3 Experiments with memetic algorithms

Due to its success and efficiency in the simulated annealing experiments, we focused exclusively on the sum of squares in the DDT as the basis for the fitness function. In all the below experiments, we carried out 100 runs of the memetic algorithm.

The first set of experiments varied the crossover method and selection method, as well as the *max_possible_mutations* and *mutation_probability* criteria. The population size was set to 400, crossover probability to 1, *elitism_level* to 1, and *no_of_children* to 2:

During earlier experiments with genetic algorithms (memetic algorithms without a hill-climbing stage), there had been reason to believe that, depending on the population size, a certain value of *max_possible_mutations* $\times$ *mutation_probability* would prove to be optimal. In the above experiments, there is far too much variation among the results to draw any such conclusion. However, it is clear from Tables 5.6, 5.7, 5.8 and 5.9 that the combination of cycle crossover and rank selection performs much more poorly than the other three (crossover, selection) choices. Furthermore, the combination of PMX and rank selection has led to higher percentages (14, 15, 16) of APN S-boxes than any of the other combinations, so we opted to stick with this for the second set of experiments. Choosing *max_possible_mutations* and *mutation_probability* was similarly difficult due to the extent to which the results varied - we eventually opted for 1 mutation with probability 0.6.

We compared the results of imposing restrictions based on Theorem 5.1.18 with the

	$max\_possible\_mutations\ (m) \times mutation\_probability$									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$m = 1$	(9, 8)	(6, 4)	(5, 3)	(6, 2)	(5, 5)	(4, 4)	(12, 7)	(7, 2)	(2, 0)	(11, 6)
$m = 2$	(12, 8)	(7, 5)	(5, 3)	(8, 6)	(7, 6)	(5, 4)	(10, 5)	(9, 4)	(6, 5)	(8, 4)

Table 5.6: (Percentage of DU 2, percentage of $(DU\ 2, NL\ 12)$) for $n = 5$ with cycle crossover and roulette-wheel selection. $(DU\ 2, NL\ 12)$ is the theoretical optimum with regard to our search criteria.

	$max\_possible\_mutations\ (m) \times mutation\_probability$									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$m = 1$	(2, 0)	(1, 1)	(1, 1)	(3, 3)	(3, 3)	(3, 0)	(4, 3)	(0, 0)	(3, 2)	(1, 1)
$m = 2$	(2, 1)	(0, 0)	(2, 1)	(2, 2)	(2, 2)	(2, 1)	(2, 1)	(2, 0)	(3, 2)	(6, 1)

Table 5.7: (Percentage of DU 2, percentage of $(DU\ 2, NL\ 12)$) for $n = 5$ with cycle crossover and rank selection.

	$max\_possible\_mutations\ (m) \times mutation\_probability$									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$m = 1$	(11, 6)	(6, 4)	(9, 7)	(9, 4)	(13, 8)	(11, 5)	(8, 4)	(8, 6)	(12, 7)	(6, 3)
$m = 2$	(4, 4)	(6, 3)	(12, 6)	(11, 7)	(7, 4)	(6, 4)	(9, 7)	(9, 2)	(10, 5)	(3, 2)

Table 5.8: (Percentage of DU 2, percentage of $(DU\ 2, NL\ 12)$) for $n = 5$ with PMX crossover and roulette wheel selection.

	$max\_possible\_mutations\ (m) \times mutation\_probability$									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$m = 1$	(14, 8)	(6, 4)	(8, 8)	(9, 8)	(9, 4)	(16, 10)	(14, 5)	(9, 7)	(16, 10)	(12, 5)
$m = 2$	(15, 11)	(8, 5)	(9, 3)	(14, 7)	(11, 5)	(8, 6)	(12, 9)	(5, 2)	(16, 8)	(14, 8)

Table 5.9: (Percentage of DU 2, percentage of $(DU\ 2, NL\ 12)$) for $n = 5$ with PMX crossover and rank selection.

results of not doing so (as also of allowing the solution candidates to make moves during mutation/hill-climbing that would violate these constraints and then retransform.) However, for every set of parameters for which this was tried, it resulted in worsened performance. We did not therefore make use of Theorem 5.1.18 in the experiments which followed.

Using these parameters, we experimented with varying the crossover probability. As part of this, we re-ran the original (1 mutation, probability 0.6, crossover probability 1.0) experiment:

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	0	0	0	0	0	2	3	8	15

Table 5.10: Percentage of DU 2 for various crossover probabilities. Again, $n = 5$.

As may be seen, the performance of the memetic search drops markedly as crossover probability is reduced. For this reason, we kept this fixed to 1.0 for the third set of experiments, in which we varied the size of the population.

Population size	% DU 2	% (DU 2, NL 12)	Time taken (d:h:m:s)
200	1	1	00:01:46:01
400	15	7	00:03:44:21
800	18	11	00:07:27:48
1600	44	22	00:15:06:30
3200	66	37	01:06:14:15

Table 5.11: Memetic algorithm results for various population sizes with $n = 5$. All experiments were carried out on a computer with a 2.66GHZ Intel Xeon X5355 processor.

It may be seen from Table 5.11 that the quality of the solutions increased with the size of the population, although the time required to obtain these solutions also increased. Nevertheless, the experiment with population size 3200 outperformed (66% DU 2 instead of 45%) the simulated annealing experiment with 30,000,000 inner loops in Table 5.3; and both of these experiments required roughly the same amount of time.

5.4 Experiments with ant colony optimisation

In Section 3.5.4 of this thesis, we stated:

"*The problem should also allow a useful cost function to be devised such that, during the*

construction of each candidate solution, the cost starts at zero and is increased whenever a new component is added until the final cost is derived."

In this case, basing the cost function on the DDT allowed us to do this; we could, for each $S(i)$ that was to be assigned a value, calculate for each j which values in the DDT would be increased, and could calculate what the new cost would be if the resultant DDT values were input directly to the cost function. As with the memetic algorithms, we used the sum of squared DDT entries as a cost function; with each new truth table entry assigned a value, we could deduce which DDT entries would increase by 2 and how this would affect the sum of squares.

We made the following decisions with regard to some of the other parameters:

- Early experiments indicated that setting $hillclimb\_trails$ to false always led to worse results, so we fixed it at true in all of the experiments in this chapter.
- We used control values $\alpha = 1$, $\beta = 2$.
- When experimenting with values of e, the values 0.05, 0.1, 0.2, 0.3 and 0.4 were tried, with 0.1 as control value in accordance with various significant works on ant algorithms [23] [119].
- Based on calculations of the optimal cost, and advice in the existing literature [23], we set τ_0, the initial amount of pheromone on each edge, to $1.0/(2^n * ((2^n - 1) * 2^n)/2.0)$.
- The control value for the number of ants was 10 in accordance with the arguments in Dorigo et al.'s seminal paper on ACS [119].
- For the problem size $n = 5$, we experimented with the values 1, 10, 100, and $(2^n * (2^n - 1))/2 = 496$ for Q.
- In the experiments with ACS algorithms, the control value for q_0 was 0.98, with 0, 0.1, 0.25, 0.5, and 0.75 also being tried. For Ant System, q_0 was of course always zero.
- The control value for ρ was 0.1, again in accordance with various papers on ACS [23] [119]. 0.05, 0.2, 0.3, 0.4, and 0.5 were also tried.

As with the memetic algorithms, we used the sum of DDT squares (and the cumulative effect of each added edge on it) as a cost function. The first major set of experiments varied Q, q_0, and the ant method:

Q	1	10	100	496
(% DU 2) (cycle)	9	5	9	6
(% DU 2) (increment)	11	6	7	8

Table 5.12: (% DU 2) for $n = 5$ with Ant System

	Q			
	1	10	100	496
$q_0 = 0$	7	5	12	2
$q_0 = 0.1$	3	3	6	8
$q_0 = 0.25$	9	4	7	3
$q_0 = 0.5$	6	7	5	2
$q_0 = 0.75$	1	1	4	3
$q_0 = 0.98$	1	1	4	3

Table 5.13: (% DU 2) for $n = 5$ with various values of Q and q_0 for Dorigo ACS (cycle).

	Q			
	1	10	100	496
$q_0 = 0$	9	1	2	5
$q_0 = 0.1$	8	5	3	2
$q_0 = 0.25$	10	3	3	1
$q_0 = 0.5$	2	2	2	1
$q_0 = 0.75$	6	1	6	0
$q_0 = 0.98$	1	0	2	0

Table 5.14: (% DU 2) for $n = 5$ with various values of Q and q_0 for Dorigo ACS (increment).

	Q			
	1	10	100	496
$q_0 = 0$	7	10	5	6
$q_0 = 0.1$	4	7	8	5
$q_0 = 0.25$	4	4	7	2
$q_0 = 0.5$	5	10	8	7
$q_0 = 0.75$	13	5	9	4
$q_0 = 0.98$	5	5	3	5

Table 5.15: (% DU 2) for $n = 5$ with various values of Q and q_0 for Luke ACS (cycle)

	Q			
	1	10	100	496
$q_0 = 0$	5	11	6	10
$q_0 = 0.1$	7	1	3	3
$q_0 = 0.25$	7	5	8	12
$q_0 = 0.5$	12	3	4	10
$q_0 = 0.75$	8	5	6	7
$q_0 = 0.98$	7	10	11	4

Table 5.16: (% *DU* 2) for $n = 5$ with various values of Q and q_0 for Luke ACS (increment).

In the above tables, we show only the percentage of *DU* 2, since for this size *NL* 12 occurs only for *DU* 2 and the cost function has no effect on whether the *DU* 2 S-box is one of the *NL* 10 or *NL* 12 boxes we have seen so far.

As before, the amount of variation in the results makes them hard to interpret. However, large Q for Dorigo's ACS appears to be detrimental to its performance, and we set Q to 1 for the second set of experiments. It was difficult to draw any similar conclusion for Luke's ACS, but we set Q to 1 for future experiments with this for the sake of comparison. Likewise, too high a value of q_0 seemed detrimental to the performance of Dorigo's algorithm, so we set this to 0.

In the second set of experiments, we varied the evaporation rate ρ and elitist update parameter e:

ρ	0.05	0.1	0.2	0.3	0.4	0.5	Total
(% *DU* 2)	8	8	7	4	5	7	39

Table 5.17: (% *DU* 2) for Ant System (increment) with various evaporation rates.

ρ	0.05	0.1	0.2	0.3	0.4	0.5	Total
(% *DU* 2)	11	4	8	11	9	7	50

Table 5.18: (% *DU* 2) for Ant System (cycle) with various evaporation rates.

	Evaporation rate (ρ)						
	0.05	0.1	0.2	0.3	0.4	0.5	Total
$e = 0.05$	3	6	8	4	11	8	40
$e = 0.1$	5	8	10	8	6	4	41
$e = 0.2$	3	6	10	7	6	7	39
$e = 0.3$	9	7	11	6	9	6	48
$e = 0.4$	7	6	9	4	6	8	40
$e = 0.5$	6	4	9	7	8	6	40
Total	33	37	57	36	46	39	248

Table 5.19: (% *DU* 2) for Luke ACS (cycle) with varying e and ρ.

	Evaporation rate (ρ)						
	0.05	0.1	0.2	0.3	0.4	0.5	Total
$e = 0.05$	4	18	4	10	7	7	50
$e = 0.1$	5	9	9	4	10	7	44
$e = 0.2$	8	10	5	9	6	5	43
$e = 0.3$	4	9	4	5	3	10	35
$e = 0.4$	8	8	7	3	5	5	36
$e = 0.5$	5	7	11	7	6	6	42
Total	34	61	40	38	37	40	250

Table 5.20: (% *DU* 2) for Luke ACS (increment) with varying e and ρ.

	Evaporation rate (ρ)						
	0.05	0.1	0.2	0.3	0.4	0.5	Total
$e = 0.05$	9	6	8	5	8	6	42
$e = 0.1$	9	10	6	3	7	8	43
$e = 0.2$	3	5	15	8	6	5	42
$e = 0.3$	7	5	9	7	6	6	40
$e = 0.4$	3	5	8	5	6	7	34
$e = 0.5$	3	4	7	7	7	4	32
Total	34	35	53	35	40	36	233

Table 5.21: (% *DU* 2) for Dorigo ACS (cycle) with varying e and ρ.

	Evaporation rate (ρ)						
	0.05	0.1	0.2	0.3	0.4	0.5	Total
$e = 0.05$	7	5	3	6	8	6	35
$e = 0.1$	7	8	7	9	6	7	44
$e = 0.2$	7	4	10	4	9	6	40
$e = 0.3$	2	6	7	6	10	4	35
$e = 0.4$	4	4	3	6	7	6	30
$e = 0.5$	6	2	7	7	6	5	33
Total	33	29	37	38	46	34	217

Table 5.22: (% *DU* 2) for Dorigo ACS (increment) with varying e and ρ.

The above tables show a similar level of variation in the achieved results to their predecessors, with no apparent patterns. In an attempt to obtain some information on the merits of the various techniques and parameter choices, we have summed the numbers of APN S-boxes achieved across all experiments for each algorithm. Again, these results have a great deal of variance, but it does appear that Dorigo ACS with increment index is underperforming compared to the other ACS variants. In fact, we believe that the above tables are evidence that Luke's ACS outperforms Dorigo's for this particular problem.

While this is not so certain, we also note that high values of e do appear to impair performance for Dorigo ACS; there is a clear drop in performance for cycle index; and also for increment index with the exception of $e = 0.05$ (which we believe to be a statistical outlier). No such pattern is visible for the values of ρ tried, however.

Cycle clearly outperforms increment index for the Dorigo ACS experiments, and while the effective sample size for the Ant System experiments is smaller, cycle still outperforms increment in these experiments by a clear margin. This is not the case for the Luke ACS experiments, however the number of APNs in the increment-index experiments would be reduced below the number for cycle-index by a slight margin if the statistical outlier for $e = 0.05,\ \rho = 0.1$ were disregarded. We therefore believe that while this differs by ant algorithm, cycle-index is in general more effective than increment-index for this problem.

For our third set of experiments (in which the number of ants was varied), we decided to conduct the experiments for both Luke ACS and Ant System, since it was not possible to draw a firm conclusion from the evidence so far as to which was the more effective. We set ρ to 0.1 and (for Luke ACS) e to 0.1. We also set q_0 to 0:

	Number of ants			
	10	32	64	128
Ant System	6	17	40	68
ACS (Luke)	7	22	37	57

Table 5.23: (% DU 2) for varying numbers of ants.

The Ant System result, as well as outperforming the best run with the memetic algorithm, also did so in just over 19 hours as opposed to just over 34 for the memetic.

(As an aside, we had also experimented again with restricting S-box output values in accordance with Theorem 5.1.18, believing that these restrictions would be necessary (though perhaps insufficient) for the search to succeed. Ant algorithms reward edges that feature in "good" solutions by increasing the amount of pheromone on them, but any given edge $(i, S(i) = j)$, could feature in any S-box with any set of properties - it would always be possible to make an affine transformation that ensured this was so - and we hoped that the truth table restrictions imposed in Theorem 5.1.18 would overcome this. Unfortunately, this did not turn out to be the case - as with the memetic algorithms, it resulted in poorer-quality solutions on average for all the ant algorithm variants and all values of Q/q_0 tried - and we had to abandon this line of inquiry.)

5.5 Conclusions, and directions for future research.

We have demonstrated that where the criteria are differential uniformity and nonlinearity - the two most important criteria for block cipher S-boxes - that metaheuristic search is capable of matching the best theoretical results for S-boxes of size 5×5 and smaller. Unfortunately, the significant increase in the size of the search space for higher n means that the difficulty level of the problem increases extremely rapidly, regardless of the metaheuristic used, and we were not able to achieve the same success for any $n \geq 6$.

Experimenting with various cost functions, we have found a particularly fast and effective cost function for this particular problem. We have also experimented with various parameter choices and found particularly effective parameters for three different types of metaheuristic applied to this problem. In comparing the metaheuristics, we have observed that ant algorithms - and in particular Ant System - appear to be more effective than either memetic algorithms or simulated annealing. This is also the first time ant algorithms and memetic algorithms have been applied to the S-box problem.

It seems unlikely that evolutionary methods acting on the truth table alone will be sufficient to find almost perfect nonlinear (or DU 4 with low DF) S-boxes for $n = 6$ or higher. In future research, it may prove beneficial to try alternate representations of the S-box [84], however it is not currently known if any alternate representations with suitable search landscapes exist.

Further research focusing on the use of theoretical results to impose further constraints on the truth table values (and hence the search space) may also yield a breakthrough. Currently, the restrictions imposed by Theorem 5.1.18 act on a fraction of truth table entries that decreases exponentially with n, and leave a great deal of latitude for the remaining values.

Furthermore, Theorem 5.1.18 only focuses on affine equivalence due to the difficulty in finding EA and CCZ transformations that achieved the results desired while preserving bijectivity. Attempting to find such transformations may achieve the stronger restrictions required, especially given that replacing a bijective S-box with its inverse is a CCZ transformation that preserves bijectivity.

Finally, the cost of a given edge in the ant colony experiments varied depending on the other edges added beforehand. This may indicate that a version of ant colony optimization designed for dynamic problems, such as AntNet, might be possible to adapt to obtain superior performance.

Chapter 6

Nonlinear cryptanalysis and metaheuristic search for S-box approximations.

In this chapter, unlike the previous two chapters, we apply metaheuristic search - in particular, simulated annealing - to cryptanalysis. However, there has been little prior research into nonlinear cryptanalysis, and so we also design algorithms to exploit the "nonlinear approximations" we evolve, and work out the statistical frameworks governing the performance of these algorithms. We are then able to work out which properties, from the cryptanalyst's point of view, would be desirable in a nonlinear approximation. This enables us to design cost functions to search for approximations with these properties. We also study existing attempts to apply simulated annealing to this problem, and work out which types of move have smooth search landscapes defined. We state the results of this for various ciphers, and attempt to incorporate the evolved approximations into new attacks on the DES and Serpent ciphers, resulting in an improved cryptanalysis of 11-round Serpent. The results in this chapter have the potential to lead to improvements in the cryptanalysis of other well-known ciphers, and establish the potential for further exploitation of metaheuristics in cryptanalysis.

6.1 Introduction.

The basic linear cryptanalytic method [179, 180] has already been described in Section 3.2 of Chapter 3. There have, however, been several extensions and variations proposed since its discovery in 1993.

The use of multiple approximations was first seen, in a somewhat ad hoc way with limited scope for generalisation, in 1994 [180]. Later that same year, Kaliski and Robshaw conducted a dedicated investigation into linear cryptanalysis with multiple approximations [144], and subsequent research in the use of multiple approximations [38, 186] finally culminated in the new method known as *multidimensional* linear cryptanalysis [68, 69, 70, 71, 66, 196], used in the best cryptanalysis to date of reduced-round PRESENT [40] [67] and Serpent [5] [196].

Another research direction proposed was the generalisation of the method to make use of *non*linear approximations. That is, instead of being restricted to equations of the form $x_{a_1} \oplus x_{a_2} \oplus \ldots \oplus x_{a_i} \oplus y_{b_1} \oplus y_{b_2} \oplus \ldots \oplus y_{b_j}$ in the input bits x_i and output bits y_i of cipher components, the cryptanalyst could make use of higher-degree terms such as $x_{a_1} x_{a_3}$ - in other words, terms that needed the AND operation to be evaluated.

This was first proposed by Harpes, Kramer and Massey [130], and investigated in more depth by Knudsen and Robshaw [164], in which it was concluded that nonlinear approximations could replace linear approximations only in the first and last rounds of the distinguisher - and even then, there were problems (as described by Knudsen and Robshaw) that would not apply in the case of a purely linear approximation. One of these was the difficulty of finding the nonlinear S-box approximations; for a DES-sized 6×4 S-box, the search space for possible approximations was 2^{64} in size, increasing to 2^{256} for an AES-sized 8×8 S-box. This was handled by restricting the search to nonlinear approximations with degree below a certain threshold d; significantly reducing the size of the search space but also preventing better approximations of higher degree from being found [161].

The assumption that nonlinear approximations could only be used in the outer rounds of the distinguisher was partially challenged by Courtois [104, 105]. Courtois demonstrated that the use of nonlinear approximations was in fact possible in other rounds of a Feistel cipher, as long as each round's approximation was a bi-linear expression using no nonlinear parts that were not of the form $(L_{i_\alpha} \oplus L_{i_\beta} \oplus \ldots \oplus L_{i_\omega}) \cdot (R_{i_a} \oplus R_{i_b} \oplus \ldots \oplus R_{i_q})$ (where L_i and R_i were variables from the left and right-hand ciphertext blocks in round i respectively).

He did, however, have to accept a certain amount of *key-dependence*, in that a given bilinear approximation B could hold with lower bias for some key values than for others. His attack also strongly relied on the Feistel structure, and could not be generalised to attack SPN-based ciphers.

The first, and so far only, use of metaheuristics in the context of nonlinear cryptanalysis was the use of simulated annealing by Clark et al. [78] to evolve nonlinear approximations to the MARS S-box [49] of the form $f(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) = (y_{j_1} \oplus y_{j_2} \oplus \ldots \oplus y_{j_k})$, for use in the first round of nonlinear distinguishers. Their approach, building on similar work in the context of stream ciphers [77], found various nonlinear approximations holding with a significantly higher absolute bias (151/512) than the best-known linear approximations for the MARS S-box (84/512). However, no means of exploiting these in an attack on reduced-round MARS was known.

In this chapter, we attempt to build on the above research in the following directions:

- We look at the question of which moves have a smooth search landscape defined when they are used as the move function in a local optimisation-based metaheuristic for finding nonlinear S-box approximations. We adapt the metaheuristic search method of Clark et al. to prioritise these over the other move types previously defined.

- The cryptanalyst does not know the values of the key bits xored with the bits involved in the nonlinear approximation. Where n_0 denotes the nonlinear function involved, computing n_0 on the bits exposed through partial encryption/decryption means that the cryptanalyst is in fact computing $n_{\alpha_1\alpha_2\ldots\alpha_l} = n_0(x_1 \oplus k_{\alpha_1}, x_2 \oplus k_{\alpha_2}, \ldots, x_l \oplus k_{\alpha_l})$. There exist 2^l candidates for the correct function, n_i, to compute on these bits, and the cryptanalyst does not know which is correct. Furthermore, the incorrect functions may still define approximations with nonzero bias, and hence

  - may contain additional information that would be of use even if the cryptanalyst *did* know the correct function.
  - may not be possible to distinguish from the correct approximation if their biases are too close to each other.

  We devise various statistical frameworks for nonlinear modifications of the basic linear cryptanalytic algorithm - Matsui's "Algorithm 2" [179] - which can succeed in spite of - or even with the assistance of - these "related" functions (or, equivalently,

the "related approximations" they define). We describe the new attack, and how to incorporate recent advances in linear cryptanalysis into it. We also adapt the metaheuristic search algorithm to take into account the properties of the related functions. Finally, we present newly-obtained nonlinear approximations for the S-boxes of various ciphers, with bias in excess of the best linear approximations for the same, and utilise these in new attacks on reduced-round Serpent.

Figure 6.1 below depicts a 1R nonlinear approximation, which we have used successfully in attacking the Heys toy cipher [140]. The $(r-1)$-round approximation is composed of an $(r-2)$-round linear approximation, followed by a nonlinear approximation which replaces the linear approximation to round $(r-1)$. As explained above, the cryptanalyst cannot simply guess the bits from the final round key, but is also forced to deal with the incorrect approximations (and one correct approximation) derived from guessing the involved values of the penultimate round key bits. Note in particular the "partial decryption" in the final round, where twelve ciphertext bits are xored with twelve guessed key bits, and the inverse S-box is applied to each of the 4-bit results. In effect, the first part of the decryption procedure is applied to part of the ciphertext.

This chapter is structured as follows: The remainder of this section provides a brief description of linear cryptanalysis, as well as an important refinement to it due to Collard et al. [100]. Section 6.2 discusses the ways in which using nonlinear approximations affects the attack. Section 6.3 describes the new attack, including the adaptation of Collard et al.'s improved methodology to the nonlinear domain and the Feistel structure, and addresses the question of how its complexity is to be calculated.

At this point in the paper, we will have described a well-defined scenario in which we can use the evolved nonlinear approximations, and will have addressed in detail the question of how the related approximations can and should be handled. This means that we will finally be in a position where we can construct cost functions taking all this into account, and so in Section 6.4 we will describe our experiments with the new simulated annealing algorithm; including how it differs from that originally used by Clark et al. [78].

A large proportion of Section 6.3 is focused on the Data Encryption Standard, as the best-known example of a Feistel cipher. Section 6.5 discusses the application of the new technique to the DES further. It also discusses the application of the new technique to other ciphers. In particular, we:

- Describe new attacks on reduced-round Serpent,

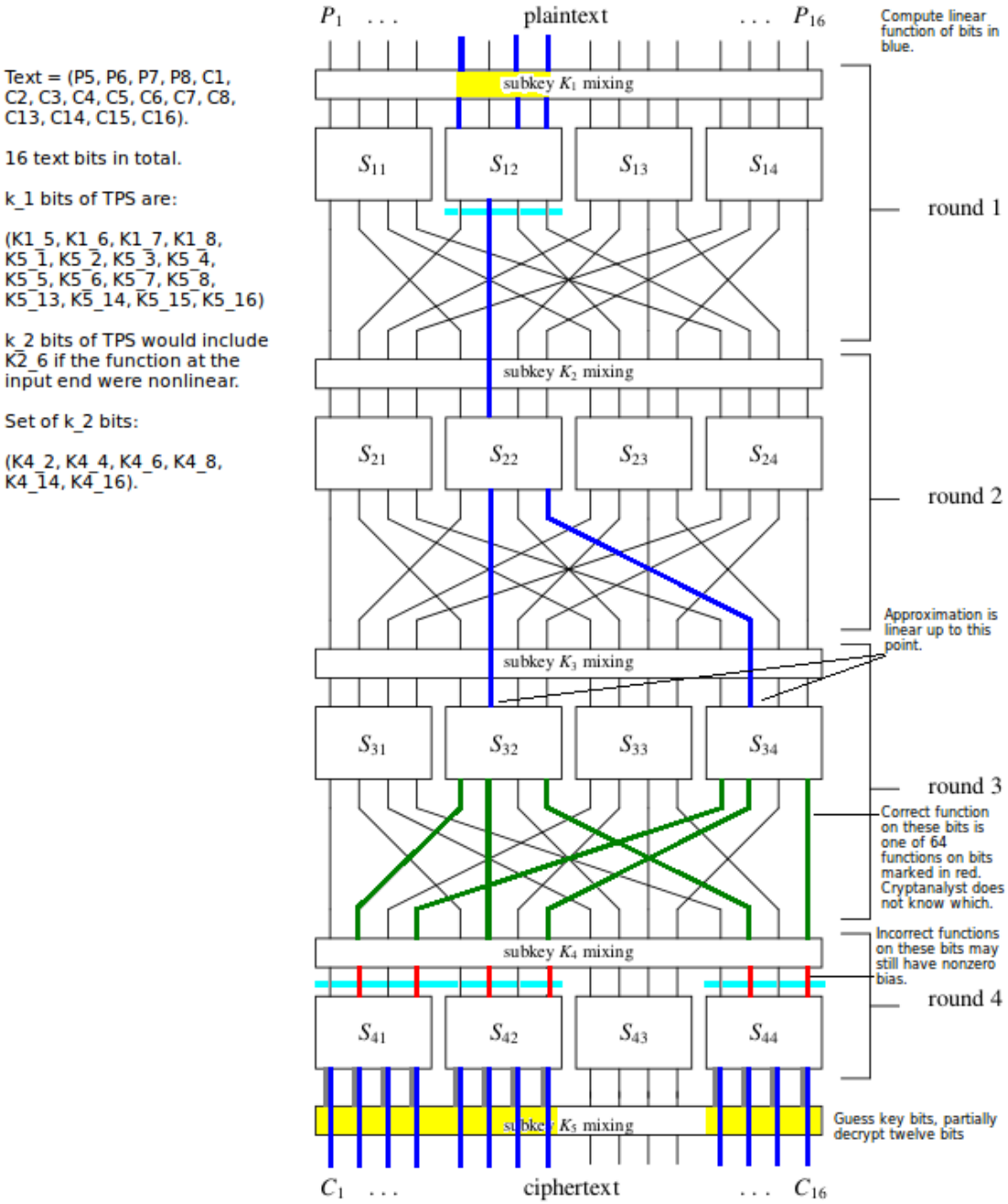

Figure 6.1: Diagram of a 1R nonlinear approximation to the Heys toy cipher [140].

- Give the results of our search for S-box approximations for the AES, DES and PRESENT S-boxes,
- Demonstrate the workings of an attack on the DES using the best of the new approximations. The attack is not in practice as efficient as the best-known attack against DES [180]; it is presented chiefly to demonstrate how a nonlinear attack on a Feistel cipher would work.

Finally, Section 6.6 presents our conclusions, and discusses avenues for further research.

6.1.1 Linear cryptanalysis - the three main phases of an Algorithm 2 attack.

Unless otherwise stated, the linear cryptanalytic attack will be assumed to be a 2R attack, in which the cryptanalyst knows of a linear approximation to rounds $2, 3, \ldots, (r-1)$ of the cipher, and by using candidate key bit values to partially decipher parts of the known ciphertexts (reversing the effects of round r on certain key bits), as well as to partially encrypt certain bits in the known plaintexts, obtains values for the bits on which the probabilistic linear relation should hold. Note in particular that, unlike the 1R attack shown in the earlier diagram, this means that key bits are guessed in both round 1 and round r.

The theoretical bias for this linear approximation is calculated by starting with the biases of the linear approximations to each individual S-box, and then using the Piling-Up Lemma [179]. This lemma was already given as Lemma 3.2.2, we restate it here for clarity:

Definition 6.1.1. For $1 \leq i \leq n$, let X_i be independent Bernoulli random variables such that:

$$p_i = P(X_i = 0)$$
$$(1 - p_i) = P(X_i = 1)$$

(In the case of linear cryptanalysis, $X_i = 0$ iff the linear approximation to the ith approximated S-box holds.)

Then $P(X_1 \oplus X_2 \oplus \ldots X_n = 0)$ is:

$$(1/2) + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2).$$

with probability bias:

$$\epsilon = 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2)$$

In reality, the probabilities of the linear approximations to the S-boxes in one round holding are not independent of the probabilities of the linear approximations to other rounds holding, so the Piling-Up Lemma only yields an estimate of the true bias. This is usually accurate enough for the purposes of cryptanalysis, although situations where it is not are discussed by Murphy [187] and Leander [172].

Definition 6.1.2. Where a linear approximation holds with bias ϵ, i.e. with probability $1/2 + \epsilon$, the *capacity* of the approximation is equal to $4 \times \epsilon^2$. More generally, in an attack using multiple approximations A_i $(1 \leq i \leq M)$, each with bias ϵ_i, the capacity of the set of the approximations is $4 \sum_{i=1}^{M} \epsilon_i^2$.

We note in particular that the bias of the linear approximation $x_{a_1} \oplus x_{a_2} \oplus \ldots x_{a_n} = y_{b_1} \oplus \ldots \oplus y_{b_m}$ as calculated using the Piling-Up Lemma may be either positive or negative, and that the values of various bits in the round keys affecting the approximated rounds may cause the actual bias to possess the opposite sign. In the standard attack, the cryptanalyst is only interested in the magnitude of the bias and hence this is not a problem. In fact, this phenomenon is actively exploited if the cryptanalyst goes on to use Matsui's Algorithm 1 [179] to obtain the parity of these key bits.

Section 3.2 of the literature review chapter in this thesis contains a description of the linear cryptanalytic attack. We propose here to build on the description by describing the way in which a linear attack is split into three main phases, and explaining in detail what the cryptanalyst does during each phase. We will then describe a recent methodological improvement to one of these phases that significantly reduces the attack's complexity, but that was felt to be beyond the scope of the introductory Section 3.2. When first published, this improvement only applied to substitution-permutation networks; we will however describe a way in which it can be adapted to Feistel ciphers such as the Data Encryption Standard. (This will prove to be important; a nonlinear attack on DES is presented in this chapter which would otherwise have overly high time complexity.)

A linear cryptanalytic attack may be divided into three main phases; each of which we need to calculate the complexity of separately. These are:

1. The *distillation phase*. In this phase, the cryptanalyst has access to N pairs (plaintext, corresponding ciphertext), all encrypted with the same key k_0. These are "known plaintext" pairs, as opposed to "chosen plaintext", because the cryptanalyst is not assumed to have been able to make any choices regarding any of the N plaintext values encrypted. The cryptanalyst needs to extract the relevant data from these pairs and discard the rest.

   Certain bit positions in the plaintext and ciphertext will have been identified as relevant, in that they are the bits which must be partially encrypted/decrypted to obtain the values of the bits involved in the approximation. Let the number of such positions be denoted l. The cryptanalyst allocates memory for an array $COUNTERS\_1$ of 2^l integer variables, each of which must be capable of holding any integer between 0 and N, and initialises these to 0. These variables are the first of several sets of counters used in the attack.

   For each known plaintext/ciphertext pair in turn, the cryptanalyst extracts the l relevant data bits. Where j denotes the l-bit value corresponding to the values of the l bits, the cryptanalyst increments the value in $COUNTERS\_1[j]$ by 1, discards the current pair, and moves on to the next pair until all N pairs have been processed.

   Clearly this phase has complexity $O(N)$. Let ϵ denote the bias of the linear approximation, then the cryptanalyst needs N to be equal (for some a) to a/ϵ^2. Advice on the value of a to choose to achieve a desired success probability was provided in Matsui's original paper [179], and later updated with a more accurate statistical framework by Selçuk [213].

2. The *analysis phase*. We shall refer to the set of key bits which are to be recovered as the *target partial subkey* (TPS). Let k denote the number of bits therein. For most ciphers, k and l will be equal; however DES's expansion phase makes it an example of a cipher for which this is not the case.

   The cryptanalyst allocates memory for an array of integers, $COUNTERS\_2$, with 2^k entries, such that each array entry should be able to take any value between 0 and N. She then, for every possible TPS value i, uses it to partially encrypt/decrypt every

possible value j of the relevant text bits in turn. If the linear approximation holds for the pair (i, j), $COUNTERS\_2[i]$ is incremented by the value in $COUNTERS\_1[j]$.

When this process is complete, the values in $COUNTERS\_2$ should be converted into the absolute values of the biases with which the approximation held for the various key guesses; this is done by mapping each value $COUNTERS\_2[i]$ to $v[i] = |COUNTERS\_2[i] - N/2|$. The higher the value of $v[i]$, the more likely it is that i is the correct TPS.

This phase as described has $O(2^{k+l})$ (usually 2^{2k}) time complexity, in that this many partial encryptions/decryptions must be carried out, each potentially requiring data to be written to an array in memory. However, this is the phase for which the aforementioned improvement exists, which we will soon address.

3. The *search phase*. During this, the correct value of the TPS bits must be obtained from the counter values calculated in the analysis phase, and the remaining key bits must also be found.

   It may be that the cryptanalyst will just accept the highest value in $COUNTERS\_2$ as corresponding to the correct key guess. The correct key guess should have reversed the effect of the outer rounds and yielded bits for which this high bias was expected; the wrong key guesses, by contrast, would not have resulted in the data bits being mapped to such values and would in effect have applied a function with a randomizing effect to them.

   If the cryptanalyst proceeds thus, various formulae exist in terms of the approximation's bias ϵ [179, 213] which can be used to calculate the number of known plaintexts N required for the attack to succeed with probability p. The cryptanalyst needs $O(2^k)$ time to search the array v for the highest value therein. After this, where K denotes the key size of the cipher, the cryptanalyst is faced with the problem of finding the remaining $(K - k)$ key bits, requiring an exhaustive search (time complexity $O(2^{K-k})$ encryptions), unless further attacks (whether linear with another approximation, or some other technique) can be applied to recovering some or all of these bits.

   However, this is not always the strategy employed. The number of possible values for the TPS bits involved is likely to be extremely high, and some of these will result, by pure chance, in high biases themselves (rather than the expected near-zero bias).

If the number of known plaintexts N does not provide a sufficiently large sample, some of these biases may be more extreme than that for the correct TPS.

In Matsui's attack on the full DES [180], this behaviour was predicted, and dealt with in a way that allowed a much lower value of N to be used than would otherwise have been the case. The correct key was expected to result in one of the X highest-ranking values of $v[i]$ (in this case, X was equal to 2^{13}), but not necessarily the highest such value. With this as the goal of the previous phases, the data complexity was much lower than it would have been had the correct key been required to yield the highest value of $v[i]$. However, this came at the cost of increased time complexity, as the search for the remaining 2^{K-k} key bits was repeated for up to X different TPS candidates.

This technique is known as *key ranking*.

The complexity of sorting the vector v to identify the highest biases is $O(k2^k))$. It may, for small values of X, be faster simply to search for the X highest biases, in at most $O(X2^k)$ time. In either case, starting with the highest-ranked TPS candidate, and continuing on for each successive candidate until the right one is found among the X highest ranked keys, the cryptanalyst must (probably through exhaustive search), search for a value for the remaining $(K-k)$ key bits such that the full key value resulting correctly decrypts the known ciphertexts.

Without key ranking, this stage should be presumed to have complexity $O(2^k+2^{K-k})$ unless there is reason (such as another high-bias linear approximation involving the other key bits) to believe that the remaining key bits can be obtained without exhaustive search. With key ranking, this stage has a higher complexity of $O(min(k,X)\cdot 2^k + X\cdot 2^{K-k})$, although as stated the use of key ranking will probably reduce the number of known plaintexts needed and hence reduce the time complexity of the distillation phase and the data complexity of the attack as a whole.

The improved method for the analysis phase, due to Collard, Standaert and Quisquater.

(Note: we now switch from referring to the ith element of an array a as $a[i]$, and will henceforth use the notation a_i.)

Other methods for the analysis phase do exist, including one used by Biham, Dunkel-

man and Keller [27] to overcome a situation in which the time complexity for the naive method described above would have been infeasible. We focus here on a newer method with very significantly improved time complexity due to Collard, Standaert and Quisquater [100]; originally defined for 2R linear attacks, and later adapted to 1R attacks by Nguyen, Wu and Wang [196].

(In both cases, the method applied only to SPN ciphers like AES and Serpent where the number of active key bits was equal to the number of active text bits (let k denote this number), and not to Feistel ciphers such as DES. In the particular case of DES, the expansion phase of the round function was another factor inhibiting compatibility, in addition to the Feistel structure. We will describe an adaptation of the method that overcomes these obstacles later on in this chapter.)

In this method (in the notation of Collard et al.):

- N, as stated earlier, is the number of known plaintext/ciphertext pairs.
- k, as also stated earlier, is the number of key bits in the TPS. It was originally assumed [100] that this was also the number l of data bits that had to be partially encrypted/decrypted, since for an SPN cipher each of these key bits would be xored with its corresponding data bit during said process. In the interests of simplicity, we will limit ourselves for the time being to ciphers such that this assumption is valid.
- C is a $2^k \times 2^k$ matrix. If the approximation holds for target partial subkey value i and value j for the relevant plaintext/ciphertext bits, $C_{ij} = 1$. If not, $C_{ij} = -1$.
- Where the "active" text bits are those which we partially encrypt/decrypt during the attack, x is a vector such that x_j is the number of (plaintext, ciphertext) pairs in which the k-bit number represented by these bits is j. Note that x is the vector we previously referred to as $COUNTERS\_1$; the computation of x is in fact the distillation phase and has complexity $O(N)$ as previously stated.
- In the case of a 1R attack, T is calculated during the distillation phase and replaces x in the algorithm. T_j is defined as (the number of P/C-pairs such that the value of the active ciphertext bits is j and the parity of the active plaintext bits is 0) - (number of pairs such that the active ciphertext bits have value j and the active plaintext bits have parity 1.)

The matrix/vector product Cx, when all entries within are divided by 2, is the previously-defined vector v such that v_i is the sample bias for TPS candidate i. We do not need to carry out this division, as the values currently present (v_i = the number of pairs such that the approximation held for candidate i, minus the number such that it did not hold.) suffice equally well. For this reason, we will engage in a minor abuse of notation and refer to Cx as v from here on. Where key ranking is involved, this vector would need to be sorted (in $O(k \cdot 2^k)$ time); otherwise it would need to be searched (in $O(2^k)$ time) for the maximum absolute value therein.

To compute and store the entire matrix C would require $O(2^{2k})$ time and memory, in addition to the $O(2^{2k})$ time complexity of the multiplication Cx. However, by relying on various properties of C, and on the Fast Fourier Transform, we are able to derive the vector $v = Cx$ using only one column of C. We can do this with time complexity $O(2^k)$ to calculate the column of C, $O(3 \cdot k \cdot 2^k)$ to compute the transforms, and $O(2^k)$ memory since only one column of C is needed for the new technique.

This is a significant improvement on the $O(2^{2k})$ complexity of the original algorithm for this phase.

(The key property of the matrix C is that the value of C_{ij} is entirely dependent on $(i \oplus j)$. Any C_{ij} and C_{gh} such that $(i \oplus j) = (g \oplus h)$ will have the same value. This means that the set of values in any one column of C is the same as the set of values in any other column - just in a different order. This redundancy is the key to the complexity improvements obtained. We do not have the space to provide a full explanation here, but refer the reader to Collard et al.'s paper [100] for the full explanation.)

Let us be a little more precise with regards to the memory requirements. The column of C has 2^k entries, all -1 or 1. This implies that we need no more than 2^k bytes to store it in signed char variables. Variable types using fewer bits are unlikely to be present on any compiler, or to have the same speed of implementation.

We also need to store x. This has 2^k entries, each of which must be at least $log_2(N)$ bits in size. On a modern processor with 64-bit word size, most ciphers will require no more than 2^{k+1} words here, or 2^{k+4} bytes. We do not know of any block ciphers in widespread use with block size > 128, although pre-AES versions of Rijndael did support up to a 256-bit block.

During the calculation of Cx, two "interim" arrays, y and z, are used [100]. Based on Carlet's description [52] of a version of the FFT using finite-field arithmetic over $GF(2)^x$,

which is equivalent to both the Fast Walsh-Hadamard Transform and the k-dimensional FFT of size 2^k [166], and on our own implementation of the same, we can say with confidence that the same data types can be used for these as for x, and hence that these arrays should require 2^{k+5} bytes between them.

This gives us a memory complexity of $2^k + 2^{k+4} + 2^{k+5} \approx 2^{k+5.615}$ bytes. One of the previous arrays can presumably be reused to store Cx itself - the space for the array that stored x could be repurposed for signed instead of unsigned data, for instance.

We now address the question of what the time complexity is in terms of. Clearly the naive algorithm would require 2^{2k} partial encryption/decryptions (PEDs) to calculate C, in addition to $O(2^{2k})$ arithmetic operations (AOs) and memory accesses (MAs) to calculate Cx. The new algorithm requires 2^k PEDs to calculate the first column of C, followed by $O(3{\cdot}k{\cdot}2^k)$ memory accesses and AOs to calculate Cx.

The question of how many arithmetic operations are involved in partially encrypting/decrypting a cipher varies by cipher, attack, and implementation. It is further complicated if lookup tables are used for the S-boxes and we are forced to evaluate the complexity of an encryption or decryption in terms of memory accesses as well, since the complexity of these will vary significantly by CPU. We will later on make use of approximate complexity in terms of arithmetic operations for reduced-round versions of Serpent using the optimised bitslice implementation [5, 6].

Based on the aforementioned version of the FFT [52], we estimate $\approx (2k + 3) \cdot 2^k$ MAs per transform. (This is only an estimate, since we do not have detailed knowledge of CPU register allocation.) Where y and z denote the output arrays from the first two transforms, the dot product $y{\cdot}z$ must then be calculated, requiring 3×2^k memory accesses. Multiplying the per-transform complexity by three, and adding the complexity of the dot product and the 2^k memory accesses when the first column of C was calculated and written into memory, gives us $\approx (6k + 13) \cdot 2^k$ MAs in total. As for arithmetic operations, the calculation of the dot product requires 2^k AOs, and based on the same evidence as before we estimate $\approx (2k + 1) \cdot 2^k$ AOs per transform, giving us a total of $\approx (6k + 4) \cdot 2^k$ AOs.

This is a significant improvement over the $O(2^{2k})$ memory accesses of the original analysis phase; although in most cases that phase was able to access contiguously stored array elements in sequence (work with $COUNTERS\_2[i]$ and $COUNTERS\_1[j + 1]$ would occur immediately after work with $COUNTERS\_2[i]$ and $COUNTERS\_1[j]$ (stored at the address prior to $COUNTERS\_1[j + 1]$)) and it may be that the extent of the improve-

ment is reduced if this factor aided the CPU's cache management/location-seeking in main memory.

We note that equating complexity in terms of memory accesses to complexity in terms of partial cipher encryptions is a difficult matter [122], depending on several factors such as; whether the CPU's memory controller is on-die or off-die, whether the memory access is to L1 cache, L2 cache, higher-level cache or main memory, the instruction set of the CPU, the efficiency of physical address extension... Previous work on the cryptanalysis of reduced-round Serpent [27, 31, 30] was not always consistent in converting between the two, and assumed 3 processor cycles per memory access - which would seem to require all memory accesses to be to L1 processor cache. Estimates for the time required to access data in main memory in the event of a cache miss vary from 75 to 300 cycles, and it is not clear if this figure is likely to increase or decrease over time, as processor performance improvements increasingly rely on multiple cores and parallel execution rather than increased clock speed. In 2003, the NESSIE project [207] gave a figure of 50 cycles per encrypted byte on either the PowerPC G3 or G4 processor as the best performance for full Serpent; if we extrapolate from this to 800 cycles per block we have a worst-case estimate of 1 MA = 3/8 of a full Serpent encryption, and we do not have up-to-date figures for more recent processors to compare this to. It is becoming accepted that there is no easy means to compare complexity in terms of memory accesses to complexity in terms of cipher operations [122], and this is a problem we ourselves will encounter when discussing the performance of our attacks in a later section.

For 2R attacks, later research [196] offers a potential performance improvement, trading very slight increases in MA and AO complexity for reduced memory and PED complexities. Let l_1, l_2 be such that $(l_1 + l_2) = k$, where l_1 denotes the number of TPS bits acting on the plaintext, and l_2 the number of TPS bits acting on the ciphertext. Then instead of 2^k partial encryption/decryptions, the method need only execute 2^{l_1} partial encryptions and 2^{l_2} partial decryptions, in addition to est. $(2^{l_2} \cdot (6l_1+4) \cdot 2^{l_1} + 2^{l_1} \cdot (6l_2+4) \cdot 2^{l_2}) = (6k+8) \cdot 2^k$ arithmetic operations and est. $(2^{l_2} \cdot (6l_1 + 13) \cdot 2^{l_1} + 2^{l_1} \cdot (6l_2 + 13) \cdot 2^{l_2}) = (6k + 26) \cdot 2^k$ memory accesses. Memory complexity is also improved, since the arrays y and z need only have $2^{max(l_1,l_2)}$ entries each, reducing the total to $2^k + 2^{k+4} + 2^{max(l_1,l_2)+5} \approx 2^{k+4.087} + 2^{max(l_1,l_2)+5}$ bytes.

A generalised version of this algorithm for use in multidimensional linear attacks was also developed [196], leading to the best cryptanalytic result so far against 12-round Ser-

pent. Where m denotes the number of dimensions, the generalised algorithm requires $2^m\times$ the number of MAs and AOs for the one-dimensional case, plus the complexity of computing $2^{l_1+l_2}$ more transforms on a data set of size 2^m, to convert the experimental correlations to empirical probability distributions.

Generalising the new method to the Feistel structure - an example.

As stated, it is in some cases possible to generalise the improved analysis phase to ciphers other than substitution-permutation networks. The key fact upon which all of the new methodology relies is that $C_{ij} = f(i \oplus j)$ for key value i and text value j. Let us note that in Matsui's attack on the full DES, twelve text bits are xored with twelve key bits prior to being input to f, but the thirteenth (an xor of eight text bits) is not. Let us therefore introduce a "dummy" key bit, the value of which we know to be zero (but which we will act as though we do not know), and assume that it is xored with one of these eight bits at the start or end of the cipher (One such bit, referred to in Matsui's notation as $C_H[29]$, is not in fact suitable for this; we must use one of the others.) This allows us to treat the partial encryption/decryption in Matsui's attack as $f(i \oplus j)$, to construct a column of C containing 2^{13} entries, and indeed to carry out the rest of the attack with complexity as described above for $k = 13$. The fact that each row for (dummy bit = 1) will be equal to $-1\times$(corresponding row for dummy bit = 0, all other key bits unchanged), and hence that the matrix will be rank-deficient, will not affect the attack.

Unfortunately, the analysis phase of Matsui's attack on the full DES has the least effect on the overall complexity, and it is this phase which would be optimised by applying the above. Furthermore, the fact that $C_H[29]$ could not be xored with a dummy key bit (since it was already one of the bits xored with a real key bit) suggests that approximations for DES, and other non-SPN ciphers, may exist to which this method cannot be applied.

6.2 How nonlinear approximations affect the attack

6.2.1 How unbalanced nonlinear components in the approximation affect the attack.

Let us assume that we have set up a 2R linear or nonlinear approximation to the inner rounds of some cipher. (It will be straightforward to extrapolate the results of this subsection to the case of 1R attacks.) For a conventional linear approximation, this would be

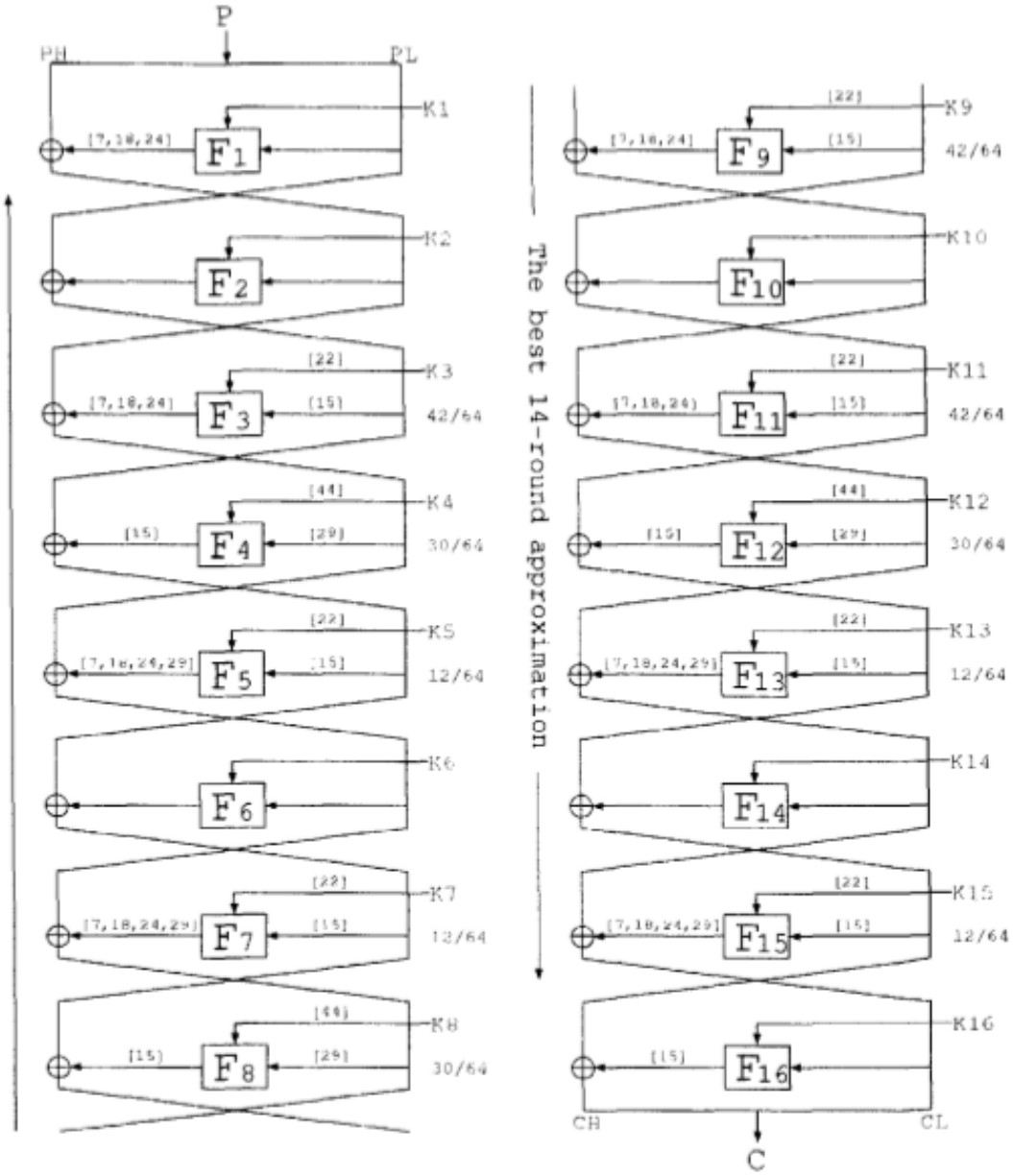

Figure 6.2: Diagram showing the full 16-round DES and the approximation for rounds 2 to 15 used by Matsui [180]. Biases are expressed as fractions, numbers in square brackets indicate active bits in the approximation.

an equation of the form $(x_{a_1} \oplus x_{a_2} \oplus \ldots \oplus x_{a_s}) \oplus (y_{b_1} \oplus \ldots \oplus y_{b_t}) = 0$, where the x_a are the input bits to Round 2 and the y_b are the output bits of round $r-1$.

Now, this equation, assuming the cipher has acted sufficiently well in randomising its outputs, should hold with bias 0 if the correct TPS has not been guessed.

We have (balanced function on one set of bits x) $\oplus$ (balanced function on some other set of bits y) $= 0$. Again, assuming the cipher's randomising effect has been adequate, the value of the first set of bits should be viewed as independent of the second. Then P(approximation $= 0$) $=$

$$\begin{aligned}
&P((x_{a_1} \oplus \ldots \oplus x_{a_s} = 0) \cap (y_{a_1} \oplus \ldots \oplus y_{a_s} = 0)) \\
&+ P((x_{a_1} \oplus \ldots \oplus x_{a_s} = 1) \cap (y_{a_1} \oplus \ldots \oplus y_{a_s} = 1)) \\
&= (0.5 \times 0.5) + (0.5 \times 0.5) \\
&= 0.5
\end{aligned}$$

This only depends on the linear function on the approximation's input bits (the linear component at the input end) and the linear function on the approximation's output bits (the linear component at the output end) being balanced, not on their being linear. Either or both of these could be replaced with a balanced nonlinear function without affecting this.

Therefore, for the following configurations for the overall approximation, the attack works as predicted by the usual probability model:

1. First per-round approximation (Round 2 of the cipher) in overall approximation is linear. Final per-round approximation (to Round $(r-2)$) is also linear.

2. First per-round approximation in overall approximation is a nonlinear approximation with a balanced nonlinear component. Final per-round approximation is linear.

3. First per-round approximation in overall approximation is linear. Final per-round approximation is a nonlinear approximation with a balanced nonlinear component.

4. First per-round approximation in overall approximation is a nonlinear approximation with a balanced nonlinear component. Final per-round approximation is also a nonlinear approximation with a balanced nonlinear component.

Now, let us assume that either the first per-round approximation, or the final per-round approximation, is an unbalanced function, and that the other is balanced. Without loss of generality, we may assume that it is the first per-round approximation, the approximation to Round 2 of the cipher, that is balanced. Let P(unbalanced component = 0) be denoted α.

Then, for an incorrect key, P(approximation = 0) =

$$
\begin{aligned}
&P((x_{a_1} \oplus \ldots \oplus x_{a_s} = 0) \cap (y_{a_1} \oplus \ldots \oplus y_{a_s} = 0)) \\
&+ P((x_{a_1} \oplus \ldots \oplus x_{a_s} = 1) \cap (y_{a_1} \oplus \ldots \oplus y_{a_s} = 1)) \\
&= (0.5 \times \alpha) + (0.5 \times (1 - \alpha)) \\
&= (0.5 \times 1.0) \\
&= 0.5
\end{aligned}
$$

We see that, as long as either the first or the last round of the approximation is a balanced function on the input bits to the inner rounds, or the output bits to said rounds, it does not matter whether the function acting on the bits at the other end is balanced. The question therefore arises: can we use approximations which are unbalanced at both ends?

Unfortunately, in general we cannot. Let β denote the probability that the nonlinear function at the input end is zero. Let γ be the probability that the nonlinear function at the output end equates to zero. Then P(approximation = 0) =

$$
\begin{aligned}
&P((x_{a_1} \oplus \ldots \oplus x_{a_s} = 0) \cap (y_{a_1} \oplus \ldots \oplus y_{a_s} = 0)) \\
&+ P((x_{a_1} \oplus \ldots \oplus x_{a_s} = 1) \cap (y_{a_1} \oplus \ldots \oplus y_{a_s} = 1)) \\
&= (\beta \times \gamma) + ((1 - \beta) \times (1 - \gamma))
\end{aligned}
$$

This is not always equal to 0.5. For example, let $\beta = 0.4$, $\gamma = 0.6$. Then the above is equal to 0.48, not 0.5. Let $\beta = \gamma = 0.1$ and the probability equates to 0.82, diverging further from 0.5. Clearly, having an unbalanced function at both ends of the approximation

is problematic, and it is for this reason that we limit ourselves to situations in which at least one end of the approximation is a balanced function on its respective set of bits.

The reader may, having noted that the related approximations previously referred to define several different functions at their respective ends of the approximation, be concerned that this will make it difficult to ensure that they are all balanced. Fortunately, all nonlinear components in a set of related approximations are balanced if and only if the primary approximation is balanced.

To prove this, let us assume without loss of generality that the correct key is an all-zeroes bitstring, and that the nonlinear component is in terms of the approximation's output bits. Consider that the nonlinear component of the $\alpha_1\alpha_2\ldots\alpha_l$th related approximation, $n_{\alpha_1\alpha_2\ldots\alpha_l}$, is equal to $n_0((y_1 \oplus \alpha_1), (y_2 \oplus \alpha_2), \ldots, (y_l \oplus \alpha_l))$, where n_0 is the nonlinear component of the correct, or primary, approximation. Clearly, each related approximation n_i must have a truth table which is a permutation of that of n_0, the permutation being determined by the fact that $n_i(y) = n_0(y \oplus i)$.

6.2.2 How the related approximations affect the attack.

We have already discussed the difficulty faced by the cryptanalyst in working out which of 2^h functions on the partially-decrypted ciphertext bits (and partially-encrypted plaintext bits) is equivalent to the nonlinear function on the S-box output/input bits involved in the approximation. One possible approach would be to compute all possible functions, and for each guess at the key bits involved, accept the function with the highest probability bias as correct.

Knudsen and Robshaw [164] considered a very simple form of this, in which no partial decryption was involved. In effect, they carried out a "0R" attack in which the whole cipher (5-round DES) was nonlinearly approximated, using an approximation that was linear on the plaintext bits but nonlinear on the ciphertext bits. The nonlinear approximation to the final round had an absolute bias of 24, and the aim of the attack was to deduce the four key bits $k_{\alpha_1} \ldots k_{\alpha_4}$ which were xored with the final-round S-box input bits involved in the approximation.

The problem that occurred was that several of the "related" functions corresponded to alternative nonlinear approximations which also possessed high bias. One of these possessed the same absolute bias as the original, and for those which did not, it was not clear how much data would be required to distinguish, say, the correct function defining a

bias 24 approximation from an incorrect function which defined a bias 16 approximation. Or, in some of the situations we encountered when devising our own approximations, a bias 24 approximation and an incorrect function defining an approximation with bias -22.

Let us try to demonstrate, using examples, why this problem does not apply in the case of linear cryptanalysis, and why attacks based on nonlinear cryptanalysis cannot disregard it in the same way as conventional linear attacks.

In a conventional 2R linear attack, key bits are guessed for S-boxes in the first and last rounds of the cipher. It is not necessary to guess the key bits affecting the S-boxes in the first and last rounds of the *approximation*. Any guess, right or wrong, at these bits simply xors a linear function with a constant value. Some key guesses will, in effect, always xor the correct function calculation with zero and leave it unaffected. Others, by always xoring with 1, will merely flip the sign of the bias.

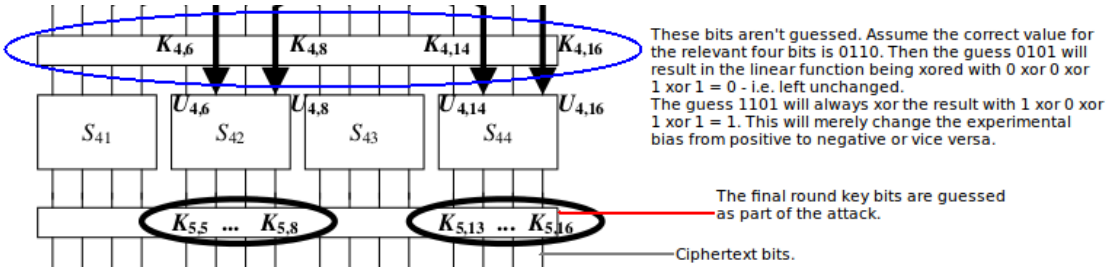

Figure 6.3: Diagram showing the final key xors and last round of the Heys toy cipher during a conventional linear attack.

In the context of Matsui's attack on 16-round DES [180], this means that although the first and last rounds of the approximation are, respectively, 2 and 14, it is not necessary to guess the round key bits which are xored with S-box input values in these rounds. Only round key bits from rounds 1 and 16 are needed.

For a nonlinear attack, this is not the case. If either the first or last round of the approximation involves a nonlinear component, and if the bits involved in said component are xored with key bits after leaving/before entering the active S-box, these key bits have to be guessed.

Let x_i denote the ith input bit to whichever S-box we are dealing with, and let y_j be the jth output bit. Let us compare the linear approximation $x_4 \oplus x_5 = y_3 \oplus y_4$ to DES S5 with the nonlinear approximations

- $x_3 \oplus x_4 = y_4 \oplus y_3 \oplus y_1 y_3$

- $x_3 \oplus x_4 = 1 \oplus y_1 \oplus y_3 \oplus y_4y_1 \oplus y_3y_4 \oplus y_3y_4y_1$

to Serpent S3:

Related approximation	Linear function	Bias
0	$x_4 \oplus x_5 = y_3 \oplus y_4$	+6
1	$(x_4 \oplus 1) \oplus x_5 = y_3 \oplus y_4$	-6
2	$x_4 \oplus (x_5 \oplus 1) = y_3 \oplus y_4$	-6
3	$(x_4 \oplus 1) \oplus (x_5 \oplus 1) = y_3 \oplus y_4$	+6

Table 6.1: Linear approximation to DES S5. Note that all relateds are either the original approximation or $1\oplus$ it.

Related approximation	Nonlinear function	Bias
0	$x_3 \oplus x_4 = y_4 \oplus y_3 \oplus y_1y_3$	+6
1	$x_3 \oplus x_4 = y_4 \oplus y_3 \oplus (y_1 \oplus 1)y_3$	0
2	$x_3 \oplus x_4 = y_4 \oplus (y_3 \oplus 1) \oplus y_1(y_3 \oplus 1)$	0
3	$x_3 \oplus x_4 = y_4 \oplus (y_3 \oplus 1) \oplus (y_1 \oplus 1)(y_3 \oplus 1)$	+2
4	$x_3 \oplus x_4 = (y_4 \oplus 1) \oplus y_3 \oplus y_1y_3$	-6
5	$x_3 \oplus x_4 = (y_4 \oplus 1) \oplus y_3 \oplus (y_1 \oplus 1)y_3$	0
6	$x_3 \oplus x_4 = (y_4 \oplus 1) \oplus (y_3 \oplus 1) \oplus y_1(y_3 \oplus 1)$	0
7	$x_3 \oplus x_4 = (y_4 \oplus 1) \oplus (y_3 \oplus 1) \oplus (y_1 \oplus 1)(y_3 \oplus 1)$	-2

Table 6.2: Nonlinear approximation to Serpent S3. In this table, the polynomial forms of the related approximations are not expanded.

Related approximation	Nonlinear function	Bias
0	$x_3 \oplus x_4 = y_4 \oplus y_3 \oplus y_1y_4 \oplus y_1y_3 \oplus y_1y_3y_4$	6
1	$x_3 \oplus x_4 = 1 \oplus y_4 \oplus y_3 \oplus y_1 \oplus y_1y_4 \oplus y_1y_3y_4$	-4
2	$x_3 \oplus x_4 = 1 \oplus y_4 \oplus y_3 \oplus y_1 \oplus y_1y_3 \oplus y_1y_3y_4$	-2
3	$x_3 \oplus x_4 = y_4 \oplus y_3 \oplus y_1 \oplus y_1y_3y_4$	2
4	$x_3 \oplus x_4 = y_3y_4 \oplus y_1y_4 \oplus y_1y_3 \oplus y_1y_3y_4$	-2
5	$x_3 \oplus x_4 = y_3 \oplus y_1 \oplus y_3y_4 \oplus y_1y_4 \oplus y_1y_3y_4$	2
6	$x_3 \oplus x_4 = y_4 \oplus y_1 \oplus y_3y_4 \oplus y_1y_3 \oplus y_1y_3y_4$	2
7	$x_3 \oplus x_4 = 1 \oplus y_4 \oplus y_3 \oplus y_1 \oplus y_3y_4 \oplus y_1y_3y_4$	-4

Table 6.3: Another nonlinear approximation to Serpent S3. In this table, the polynomial forms of the related approximations *are* expanded.

For the first nonlinear approximation, in a situation where $y_1y_3 = 1$, any wrong guess at key bits (k_1, k_3) will result in its value being wrongly calculated as 0. If $y_1y_3 = 0$, by contrast, only one of the three possible wrong guesses for (k_1, k_3) will result in its value

being calculated incorrectly. In general, an incorrect key guess will not consistently result in the wrong value being assigned to the nonlinear terms affected by it, and so will not simply leave the overall magnitude of the bias involved in the attack invariant.

It is therefore necessary to guess at the key bits involved in the first and last rounds of the approximation, as well as those involved in the first and last rounds of the cipher or reduced-round variant thereof (in a 2R-attack.), simply to be able to obtain the latter set of key bits. Since having to guess the values of these bits adds to the time complexity of the attack, we would like to obtain some information about them.

Let us look again at the approximation $x_3 \oplus x_4 = y_4 \oplus y_3 \oplus y_1 y_3$ above. The related approximations when k_4 is guessed wrongly hold with the same absolute bias as the corresponding relateds for when it is not, so unfortunately we cannot recover any information about the value of k_4. However, the related for (k_4 alone wrong) is the only related with an absolute bias near to that of the correct guess, so we should be able to recover the values of bits k_1 and k_3

Now consider $x_3 \oplus x_4 = 1 \oplus y_4 \oplus y_3 \oplus y_1 y_4 \oplus y_1 y_3 \oplus y_1 y_3 y_4$. As seen in the table above, no related approximation has as high a bias as the correct one, so in theory it should be possible to obtain information on all three key bits involved. In practice, since the relateds for (k_4 wrong) and (all three key bits wrong) both have high bias, the amount of data required to distinguish these from the correct related will be higher than that for the remainder of the attack.

For this reason, in a search for approximations to use in a straightforward generalisation of the linear attack, it would seem that the cost function should try to maximise the difference between the absolute bias of the evolved approximation, and the highest absolute bias of any of the related approximations. Since the attacker needs to obtain the key bits for the first and last rounds of the cipher, the need for the "primary" approximation to possess a high absolute bias is also important.

The above was all taken into account by Knudsen and Robshaw. However, what was not observed was that related approximations with high absolute bias may actually benefit the cryptanalyst during the search for the key bits in the cipher's outer rounds. If the correct key is guessed in these rounds, the related approximations will be expected to hold with their predicted biases; if not they will be expected to hold with bias 0. By evaluating all possible related approximations, the cryptanalyst can track the information on the biases of 2^l approximations instead of just one, and may be able to use this extra

information to boost the "signal-to-noise ratio" and reduce the data requirements of the basic attack - in effect trading increased time complexity against reduced data complexity.

Moreover, it may be that the cryptanalyst will decide only to attack the key bits in the outer rounds, basing the score for each outer-round key candidate on the best experimentally obtained bias across all of the relateds. If the primary approximation is expected to hold with a particularly high magnitude of bias, the reduced data complexity resulting from this approach may be deemed a reasonable tradeoff for the increased time complexity (compared to conventional linear) in evaluating the full set of related approximations.

6.3 New statistical frameworks and cryptanalytic techniques.

6.3.1 Adapting the new analysis phase to nonlinear cryptanalysis of substitution-permutation networks.

Where the cipher being attacked is a substitution permutation network, we will describe an adaptation of Collard et al.'s new analysis method [100], as also the improvements due to Nguyen et al. [196] to nonlinear cryptanalysis. For other cipher structures, such as Feistel ciphers, the intention is to adapt this method as far as possible - indeed, in the next subsection we will discuss adapting this method to the Data Encryption Standard.

- Let k denote the target partial subkey; i.e. the set of attacked key bits. Let k_1 be the bits of k interacting with the S-boxes in the outer rounds of the cipher (the ones which we must partially encrypt/decrypt.) Let k_2 be the bits of k interacting with the S-boxes in the outer rounds of the approximation.

- Let $f(i, j)$, where i is the value of the active text bits, and j the value of the bits of k_1 with which they are xored, be a $2^{|k_2|}$-long string of values $\in \{-1, 1\}$ defined as follows:

    1. Partially encrypt/decrypt i using j. This will yield a string of text bits entering/leaving the outer rounds of the approximation, $|k_2|$ of which are involved in the nonlinear component. Note that this string of text bits is in fact only dependent on the value $(i \oplus j)$.
    2. For each possible value μ of k_2, xor the $|k_2|$ bits mentioned above with the appropriate bits of μ, and compute the nonlinear function on these. Set the

μth entry in the string of values to -1 if the nonlinear approximation does not hold when this is done. Otherwise, set it to 1.

- The string of 1s and -1s is obtained by applying a sequence of functions to a set of bits determined entirely by the value of $(i \oplus j)$. This allows the matrix C such that $C_{ij} = f(i \oplus j)$ to be defined as before, except that C_{ij} is now a string of values instead of just one.

- Where x is the vector containing the frequency with which each value for the involved text bits has occurred, Cx can also be calculated as before, although each entry in Cx is now a $2^{|k_2|}$-string of integers.

  (To clarify, let us assume that we have $2^{|k_2|}$ matrices $C(y)$, defined by letting $C(y)_{i,j}$ be equal to the yth entry in $C_{i,j}$. We can calculate $C(y)x$ for each $C(y)$, and then Cx is the vector such that Cx_i is the string of ith entries from each of the $C(y)x$ in order: $(C(1)x_i, C(2)x_i, \ldots, C(2^{|k_2|})x_i)$.)

- So far, the memory complexity, and the time complexities in terms of arithmetic operations and memory accesses, of the corresponding stages of the linear version of this method can simply be multiplied by $2^{|k_2|}$ to obtain the complexity of the new method up to this point.

- The first problem we are faced with is choosing the correct value of j (i.e. of k_1) from this. Each string of values needs to be assigned a score such that, according to some statistical theory, the more likely a given k_1 candidate is to be correct, the higher the score assigned to its corresponding string of values.

  In conventional linear cryptanalysis using the analysis method of Collard et al., there would be only one value in this string, the absolute value of which would be the score. The complexity of going through the values in Cx and setting them to their absolute values would be at most $2^{|k_1|+1}$ memory accesses ($2^{|k_1|}$ reads, and at most $2^{|k_1|}$ writes.) and $2^{|k_1|}$ arithmetic operations. More generally, the time complexity for this phase for nonlinear cryptanalysis is at least $O(2^{|k_1|} + 2^{|k_1|+|k_2|})$ memory accesses, to access all values in all strings and to write the scores to an array.

- One way in which we could handle this would be by allocating each string of values a score equal to the maximum absolute value therein. This approach, which we shall

refer to as the *maximum-bias* approach, is the simplest possible method, and may be the best to use when one of the approximations has a bias of considerably higher magnitude than any of the other relateds. However, it does fail to make use of most of the information in each vector Cx_i.

The vector of scores should need at most (block size of cipher) bits per entry. Currently most block ciphers have block size ≤ 128, so this usually adds $\leq 16 \times 2^{|k_1|}$ bytes to the memory complexity. The time complexity will be dominated by the $O(2^{|k_1|} + 2^{|k_1|+|k_2|})$ memory accesses.

- Another possible approach would be to allocate each string of values a score equal to the sum of squares of the values therein, before either accepting the value of k_1 with the highest score or key ranking according to this score.

  The time complexity for scoring according to this method should not differ substantially from the maximum-bias method, but the memory required for the vector of scores would be substantially higher - $2^{|k_1|+|k_2|} \times 2\lceil log_2(N) \rceil$ bits, $\leq 2^{|k_1|+|k_2|} * (BLOCK\_SIZE * 2)$, since in theory an attack using the full codebook could result in at least one score equal to $2^{2 \times BLOCK\_SIZE}$. For a 128-bit block cipher, this leads to an upper bound of $\leq 2^{|k_1|+|k_2|} * 32$ bytes.

  If the truth tables of the related approximations are statistically independent, this will allow us to make use of the χ^2-statistic in a way similar (but not identical) to its use in multidimensional linear cryptanalysis [70, 66].

  If they are not, we still gain information from the sum of squares that allows them to be used as a distinguisher, but we do not gain as much as if they were independent. Since there is no known statistical framework for a variation of the χ^2-statistic where some of the Pearson correlation coefficients of the variables are not $\in \{0, \pm 1\}$ (i.e. where they are not independent), we will need to conduct experiments on significantly reduced-round cipher variants to obtain empirical evidence for the distinguishing advantage obtainable.

- The question arises as to whether we could exploit our knowledge of the theoretically predicted distribution of the biases of the nonlinear approximation and its relateds. For example, if we expect the related approximation for $k_2 \oplus \alpha$ for some value α to hold with bias β, and the related approximation for $k_2 \oplus \gamma \oplus \alpha$ (for some value γ) to hold with bias $-\beta$, the above approaches do not currently utilise this knowledge.

However, since we do not know in advance the correct value of k_2, this would require us, for each k_1, to attempt to match the distribution to every possible value for k_2 - which would result in increased time complexity.

Since the log-likelihood ratio has the optimal data complexity among all methods for distinguishing a distribution p from another distribution q [111, 70], we believe that this would be the most effective means to exploit the information referred to. Since the related approximations may not have statistically independent truth tables, though, similar problems to those described for the χ^2-statistic may still arise.

There is also the "linear hull" effect to be borne in mind. Approximations with the same input and output bitmasks, but following different paths through the cipher - i.e. different characteristics - may result in the actual distribution being different to that predicted theoretically. Figure 4 of a recent paper by Collard and Standaert [97] shows the results of experiments conducted on reduced-round versions of the cipher SmallPRESENT [171], in which this difference is seen to increase significantly with the number of rounds. In all of these experiments, we note that as the number of rounds increases, the magnitude of the theoretical bias for a conventional linear approximation (as calculated using the Piling-Up Lemma) is seen to increasingly underestimate the magnitude of the actual bias. Furthermore, the extent of this underestimate varies significantly depending on the key value, although the extent of this variation does not appear to increase further after five rounds.

For the other methods we have suggested, this would not pose a problem; indeed it would be beneficial to the attack's performance. However, for a particular distance metric (the Kullback-Leibler distance), the LLR statistic in a linear cryptanalysis variant would reward high distance from the uniform distribution, and low distance from the theoretical distribution, equally. The linear hull effect would clearly interfere with the second part of this.

Furthermore, experiments conducted on SmallPRESENT are particularly relevant to PRESENT and Serpent - in fact, SmallPRESENT is parameterisable such that for one particular parameter value, it is exactly the same as PRESENT. All three ciphers have the following in common:

– An SPN structure, so that round-key xor, application of a layer of substitution boxes to the entire block, and then a linear diffusion layer, are applied in

sequence in each round (Serpent omits the diffusion layer in the final round), followed by a final key xor at the end of the cipher.

– All the S-boxes in a given round are identical 4×4 bijections, with differential uniformity 4, nonlinearity 4, and most/all of the S-box co-ordinate functions having algebraic degree 3. In particular, the S-box used in PRESENT and SmallPRESENT is affine-equivalent to Serpent's S2 and S6.

Serpent does have a more effective diffusion layer than the permutation used by PRESENT and SmallPRESENT, but whether the increased number of active S-boxes resulting from this exacerbates the problem observed by Collard and Standaert or not is unclear - it seems extremely unlikely that it could in any way mitigate it.

Where the theoretical prediction is known to be accurate, or where experiments have indicated that it is likely to be for the particular cipher and number of rounds being attacked, the log-likelihood ratio (LLR) has been shown in the context of multidimensional linear cryptanalysis [70] to be superior to the χ^2 statistic. Approximations to the LLR statistic also exist which can be computed much more quickly - one based on its Taylor series expansion [111], another, slightly less accurate but faster to compute, based on the convolution of probability distributions [135, 136]. However, since the extent to which the linear hull effect would distort the accuracy of attack complexity calculations is currently unknown, we will not cover it in detail in this thesis.

- Upon accepting a given value of k_1, we next need to find k_2. Depending on the various parameters of the attack, there may be situations where the most practical approach is simply to include the bits of k_2 in the exhaustive search for the non-attacked key bits. For example, it may be that the incorrect key guesses result in related approximations with too high a bias to be practically distinguishable from the correct k_2 and corresponding approximation.

  As an example, let us consider an attack on Serpent in which only the final round of the approximation contains a nonlinear component; this being in S-box S3 with input bitmask 11 (so $x_1 \oplus x_3 \oplus x_4 =$ some nonlinear function of the output bits with some bias ϵ.). We have eight nonlinear approximations to this bitmask with bias 6, four of which are of particular interest here. Each of these four has one related

approximation with a bias of -6, one related approximation with bias 2, and one related approximation with bias -2.

(Approximations with these biases occur frequently for 4×4 S-boxes. They are especially useful for various reasons:

- Both of the related approximations with absolute bias 2 are statistically independent of the approximation with bias 6.
- The approximation with bias -6 has a truth table which can be obtained from the truth table of the bias 6 approximation by flipping all of the bits therein. This means that it provides no information that the approximation with bias 6 does not, and can safely be omitted from the attack.
- The approximation with bias 2 is related to the approximation with bias -2 in the same way. This means that only one of them provides useful information, and the other can be omitted from the attack. It is up to the cryptanalyst to decide which one.
- Their nonlinear components are balanced.

We are therefore able to handle statistical dependence among the related approximations in an extremely straightforward fashion, leaving us with a set of completely independent approximations for which the χ^2 statistic is fully valid, and for which the LLR statistic is also valid (barring issues resulting from the linear hull effect).)

We can use any of the approximations individually, or we can attempt a form of multiple nonlinear cryptanalysis using two or more approximations (or, equivalently, two or more sets of related approximations) simultaneously. The below pseudocode demonstrates the attack for both the one-approximation and two-approximation cases.

For each value of k_1, based on whichever statistical method is in use (whether maximum-bias, χ^2 or other) we assign a score to the distribution of values in its corresponding Cx entry. For the maximum-bias and χ^2 methods, the scoring system should reward high values for the distance between the experimentally obtained distribution and the uniform distribution. A randomly-chosen wrong key is expected to possess much lower distance than the correct key; however (as noted in subsection

Algorithm 7 Nonlinear cryptanalysis algorithm

$l \leftarrow$ the number of active data bits.
$h \leftarrow$ the length of k_2.
for $(i \oplus j) \leftarrow 0, 2^l - 1$ **do**
    Partially encrypt/decrypt $(i \oplus j)$.
    Let m denote the result of this.
    Let μ denote the bits of m involved in the nonlinear component(s).
    for $CURRENT\_K_2\_VAL \leftarrow 0, 2^h - 1$ **do**
        $\delta \leftarrow \mu \oplus CURRENT\_K_2\_VAL$
        if Attack uses one approximation **then**
            Compute nonlinear function on δ
            if Approximation holds **then**
                $C_{ij}[CURRENT\_K_2\_VAL] \leftarrow 1$
            else
                $C_{ij}[CURRENT\_K_2\_VAL] \leftarrow -1$
            end if
        else if Attack uses multiple approximations **then**
            for $CURRENT\_APPROX \leftarrow 0, NO\_OF\_APPROXIMATIONS$ **do**
                Compute current nonlinear function on δ
                if Current approximation holds **then**
                $C_{ij}[CURRENT\_APPROX][CURRENT\_K_2\_VAL] \leftarrow 1$
                else
                $C_{ij}[CURRENT\_APPROX][CURRENT\_K_2\_VAL] \leftarrow -1$
                end if
            end for
        end if
    end for
end for
Compute Cx.
We obtain, for each value of k_1, a vector of values.
We allocate a score to this vector depending on the statistical method in use.

6.1.1) this does not necessarily mean that the correct key will possess the highest distance, and some form of key-ranking may be required.

An important complexity issue arises. In determining whether the nonlinear approximation holds for each candidate k_2 value, we need to repeatedly evaluate a nonlinear expression, and the complexity of this compared to evaluating a linear expression in the conventional attack (considered negligible in most papers) is not clear (although experiments have confirmed that it is considerably higher.)

For example, this is the nonlinear component of an approximation to DES S5:

$$1 \oplus x_5 \oplus x_5x_6 \oplus x_2x_6 \oplus x_1x_5 \oplus x_1x_2 \oplus x_1x_5x_6 \oplus x_1x_2x_6$$

It is not clear how to compare the complexity of this to the complexity of the full S-box, as it is unlikely that an S-box implementation would rely solely on XOR, AND and NOT (to add the constant term) gates. Moreover, the difficulty of finding, for a given basis and function, the circuit for that function with the smallest number of gates is a difficult and still open problem [108]. It is to be assumed that the cryptanalyst would be using S-box implementations chosen to maximise speed, without regard to such factors as resistance to side-channel attacks which most cipher implementations would have to address.

Since this may be represented by a lookup table with as many elements as the S-box:

$$1101110111011101100010001000100011111111111111110000000000000000,$$

and since its algebraic normal form has a much smaller weight than any co-ordinate function of the S-box it approximates, we will proceed with the assumption that the complexity of calculating this function is less than or equal to that of computing a full S-box. Since it will have to be calculated $2^{|k_2|}$ times for each partial encryption/decryption, where S_c denotes the total number of S-boxes in all the rounds of the cipher, we will estimate the time required for each partial encryption/decryption to be (number of active outer round boxes)$/S_c + 2^{|k_2|}/S_c$ of the time required for a full encryption.

6.3.2 The theoretical complexity of the new attack.

The key question that now arises is how much the use of nonlinear approximations will affect the known-plaintext requirements for the attack. It seems at first sight that they should be significantly reduced. For example, if we use the nonlinear approximation to

DES S5 above in the final round of Matsui's attack [180], instead of one approximation to the whole cipher with bias -1.192×2^{-21}, we have a set of eight related approximations, including one with bias 1.431×2^{-21}.

(More precisely: one approximation with bias 1.431×2^{-21}, one with bias 1.907×2^{-22}, two with bias -1.907×2^{-22}, two with bias 1.431×2^{-22} and two with bias -1.431×2^{-22} - unfortunately, these are not statistically independent of each other.)

We cannot adapt the complexity predictions from Biryukov et al.'s 2004 work on linear cryptanalysis with multiple approximations [38], since Murphy [188, 189] has demonstrated flaws in the crucial Corollary 1, and shown that it underestimates the data requirements (increasingly so as the number of approximations increases.)

We have, instead, focused on adapting the data complexity predictions for multidimensional linear cryptanalysis, which uses different statistical frameworks not affected by this flaw. However, this does bring further problems to light.

Before we discuss these, we define the essential concept of *advantage*:

Definition 6.3.1. If, in a cryptanalytic attack on a TPS of length n, we are employing key ranking such that the attack will be considered a success if the correct TPS is one of the 2^{n-a} highest ranked, the value a is referred to as the *advantage* of the attack.

We use the following notation:

- a denotes the advantage.
- When discussing the χ^2 statistic, b denotes the value $\Phi^{-1}(1-2^{-a})$. When discussing the LLR, b denotes the value $\Phi^{-1}(\sqrt[M+1]{1-2^{-a}})$.
- k_1^0 denotes the correct value of k_1.
- k_2^0 denotes the correct value of k_2.
- N denotes the number of known plaintext/ciphertext pairs involved in the attack.
- In nonlinear cryptanalysis, M denotes the number of related approximations. In the case of multidimensional linear cryptanalysis, $M = 2^m - 1$ is the number of linear approximations involved in the attack; these being the nonzero linear combinations of the m base approximations.
- P_S is the success probability of the attack.

- $C(p)$ is the theoretical capacity of the set of approximations used. Since p is clearly the theoretical distribution from the context in which this is referred to, we will sometimes simply denote it C.

The chi-squared statistic.

Consider the complexity calculations for the χ^2-statistic in multidimensional linear cryptanalysis [70]. Prior to Theorem 1, the authors state that $\Phi(-b) \approx \frac{e^{-b^2/2}}{\sqrt{2\pi}}$ when b is large. This is not in fact the case - the approximation was taken from Section 4 of an earlier paper [10], but the authors of this had realised that the approximation was erroneous and published a correction [182]. The correct approximation for large b is $\frac{e^{-b^2/2}}{b\sqrt{2\pi}}$.

Theorem 1 of this paper relies on rearranging

$$N \approx \frac{2\sqrt{M}b + 4\Phi^{-2}(2P_S - 1)}{C(p)}$$

to obtain

$$b^2 \approx \frac{(NC(p) - 4\Phi^{-2}(2P_S - 1))^2}{4M} \tag{6.3.1}$$

The authors, relying upon the formula $2^{-a} = \Phi(-b)$, applied the approximation $2^{-a} = \Phi(-b) \approx \frac{e^{-b^2/2}}{\sqrt{2\pi}}$, claiming from this that $a \approx b^2$ and so that the above equation gave an approximate formula for the advantage of the attack in terms of N. Since $\sqrt{e} = 1.648721271 \neq 2$, this is already incorrect. Allowing for the correction to the approximation, $a \approx \frac{b^2}{2 \cdot ln(2)} + log_2(b) + log_2(\sqrt{2\pi}) \approx 0.72b^2 + log_2(b) + log_2(\sqrt{2\pi}) \approx 0.72b^2 + log_2(b) + 1.325$.

The value b is not so large as to allow us to simplify further with $a \approx 0.72b^2 + log_2(\sqrt{2\pi})$; this would result in a 2-bit underestimate for the advantage in Cho's attack on PRESENT [67].

We note, however, that for relatively marginal attacks with low advantage, the condition of "large b" is not satisfied. Later on in this section, we will plot graphs of the estimated advantage based on this approximation, and on the direct computation of $a = -log_2(1 - \Phi(b))$, and show that the approximation mistakenly predicts that low advantage attacks with data complexity below 2^{27} on reduced-round Serpent cannot be mounted. We recommend calculating a from b directly if possible.

However, the problems run deeper still. Despite contacting the authors of the key

paper [70], we have not been able to re-derive the approximation

$$NC(p) \approx 2\sqrt{M}b + 4\Phi^{-2}(2P_S - 1)$$

One of the authors, taking into account the incorrect $\Phi(-b)$ approximation, has stated that there may have been a mistake, but no longer has access to the software originally used in obtaining the approximation. In particular, we believe that there is no way to obtain an approximation containing $\Phi^{-2}(2P_S - 1)$, and conjecture that this results from a misunderstanding of the formula $(2P_S - 1) = erf(\frac{\Phi^{-1}(P_S)}{\sqrt{2}})$.

Equation 9 of the same paper is:

$$\Phi^{-1}(P_s) = \left(\frac{\mu_R - \mu_a}{\sqrt{\sigma_R^2 + \sigma_a^2}} \right)$$

in which:

$$\sigma_a^2 = \frac{2M}{2^{n+a}\phi(b)^2}$$

In solving Equation 9 to obtain a formula for $NC(p)$, the approximation $\sigma_a^2 \approx 0$ was originally made [70], and a quadratic equation with $NC(p)$ as the variable is formed. In a later revision [199], the approximation $\sigma_a^2 < M$ is used; and the formula for NC uses $\sigma_a^2 = M$ to provide a conservative value for NC.

As n denotes the number of key bits targeted in the attack, we argue that $n \geq 3$ (this value could in theory result during a 1R linear attack on a cipher using the CTC2 S-box [107]. A block cipher's S-boxes cannot have less than 3 input bits if they are to be nonlinear balanced functions. For an SPN the number of output bits would also have to be 3 and for a Feistel cipher it would only be the number of S-box input bits that was relevant), hence $(n+a) \geq (3+a)$. Moreover, it is clear that $(n+a) \geq 2a$. Exploiting these facts, it is possible to verify that M does represent an upper bound for advantage ≥ 1. (Using the computer algebra package Mathematica, we draw graphs of $2M/(2^{3+a}\phi(b)^2)$ and $2M/(2^{2a}\phi(b)^2)$ to confirm this.) Using information from the graphs, we were able to obtain a tighter upper-bound of $0.7854M$.

Note that depending on how close the advantage is to n, the significance of the over-estimate varies substantially. For example, $\sigma_a^2 = M/2^{5.2}$ is attained for $a = n = 56$. For a smaller advantage of 32 and $n = 56$, $\sigma_a^2 \leq M/(2^{28.35}) \ll M$. However, if M=$2^{56} - 1$ (as

used in various attacks on reduced-round Serpent [196]), this is not ≈ 0.

(Note that said attacks must in fact have used the log-likelihood ratio, not the χ^2-statistic, to succeed for so high a value of M.)

Replacing σ_a^2 with $0.7854M$ in the aforementioned quadratic equation, we use the quadratic formula to solve the equation in NC and obtain:

$$NC(p) \approx 2\Phi^{-2}(P_S) + \sqrt{2M}b \pm \sqrt{\Phi^{-2}(P_S)(4\Phi^{-2}(P_S) + 4\sqrt{2M}b + 2.7854M)} \quad (6.3.2)$$

We note that the expression under the square root sign cannot take on a negative value for $P_s > 0.5$ as long as the advantage a is greater than or equal to 1 bit - and depending on P_s may still not be negative even for extraordinarily marginal attacks with lower a. So we are able to accept that the roots of this equation will be non-complex in real-world attack situations.

The question arises as to whether the larger or the smaller of the two roots should be considered the solution. For an attack obtaining a 4-bit advantage and probability of success ≥ 0.95, the smaller root is a negative value, strongly indicating that it cannot be the correct solution. Furthermore, in email correspondence, we obtained from Nyberg [199] a pessimistic formula for NC relying on certain assumptions. If we consider situations in which these assumptions hold, we find that the smaller root diverges massively from the value given by this formula, while the larger root does not differ to such an extent. Based on this, we conclude that the smaller root does not match the true complexity of the attack and that the larger root is the correct value of NC: [1]

$$NC(p) \approx 2\Phi^{-2}(P_S) + \sqrt{2M}b + \sqrt{\Phi^{-2}(P_S)(4\Phi^{-2}(P_S) + 4\sqrt{2M}b + 2.7854M)} \quad (6.3.3)$$

Using this new model, we find b by using Mathematica to solve the equation above, after which we can either compute the advantage directly from b or use the "large b" approximation referred to earlier. For the 4, 7, 10 and 12-dimensional χ^2 attacks on 5-round Serpent in Hermelin et al.'s FSE 2009 paper [70], we use the corresponding capacities

[1] To avoid the use of computer algebra packages in deriving the above formula, it is stated [199] that $NC < M/4$ and that this should be substituted for NC in the denominator, resulting in a pessimistic estimate for NC. Although this seems to have been the case for all multidimensional linear attacks so far, M is often much smaller in nonlinear attacks, and certainly exceeded $4NC$ in the attack on DES below, so we were unable to make this substitution.

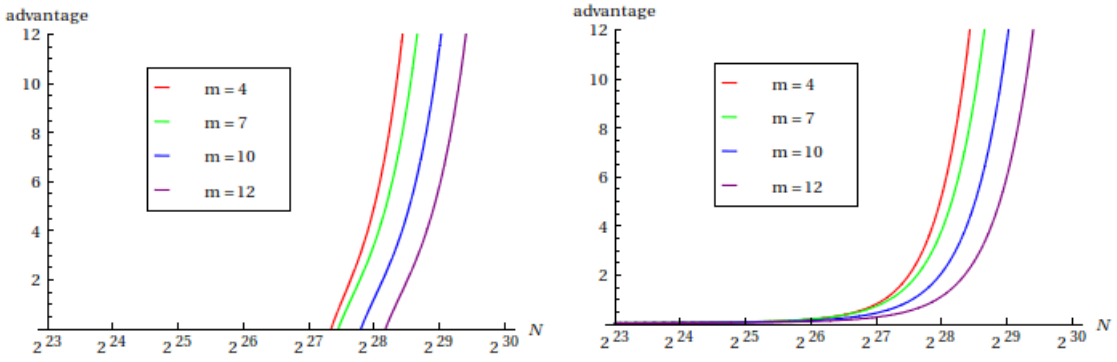

Figure 6.4: The graph on the left shows the result of using the approximation to calculate the advantage from b. We see that the approximation fails for marginal, low-advantage attacks. The graph on the right shows the result of computing advantage directly.

to plot new graphs of N against advantage (Figure 6.4).

Now, the y axes of these graphs show the advantage going up to 12, since this was the number n of bits in the TPS. However, if we do not use the graph plotting software to impose these restrictions, the graphs show the advantage continuing to increase indefinitely with an increasingly steep gradient, even though it should be upper-bounded by n. Furthermore, equation 6.3.3 does not have n as a variable after the pessimistic approximation for σ_a is introduced, suggesting underlying flaws in Selçuk's statistical model for conventional linear cryptanalysis [213] that may have been exacerbated in the generalisation to multiple dimensions.

In Selçuk's model [213], for all advantages except 0 and maximum advantage $a = n$, the rth-highest bias of any wrong key candidate ($r = 2^{n-a}$) is assumed to have an asymptotic normal distribution. Let T_1 be the lowest bias of any wrong key, T_2 the second lowest, ..., T_{2^n-1} the highest (so that T_{2^n-r} corresponds to r). A value $q \approx (1 - 2^{-a})$ $(0 < q < 1)$ is defined, such that $(2^n - r) = \lfloor q(2^n - 1) \rfloor + 1$. Since there are $2^n - 1$ wrong key candidates, we can obtain a tighter upper bound for q of $q \leq \frac{2^n-1}{2^n-2}$. As this would correspond to an attack with maximum advantage, for which the precise distribution of the highest bias of the wrong key candidate is known (assuming the Wrong-Key Randomization hypothesis), a formula based on this and not the asymptotic Normal approximation is used to calculate the attack's complexity.

(A useful topic for future research would be a generalisation of this formula to the multidimensional case, so that the effect of varying the number of dimensions on the

accuracy of the Normal approximations can be investigated.)

We can therefore say, when working with the non-extreme-value asymptotic Normal distribution, that $q < \frac{2^n-1}{2^n-2}$, and it also seems reasonable to treat the current statistical model as suspect for advantage higher than $(n-1)$.

However, there is also reason to believe that the model may not be valid for some smaller advantages $(n-x)$ either. In the textbook "Order Statistics" [115], discussing order statistic X_r (where the order statistics are $X_1 \leq X_2 \leq \ldots X_n$), David states (at the start of Section 9.1):

"If $r/n \to \lambda$ as $n \to \infty$, fundamentally different results are obtained according as $0 < \lambda < 1$ or $\lambda = 0$ or 1, with r or $(n-r)$ fixed.

"In the former case, X_r is a sample quantile and (subject to mild regularity conditions) has an asymptotic normal distribution. The latter case includes the extremes $X_1,\ X_n$ and corresponds to the mth extremes $X_m,\ X_{n-m+1}$ with m fixed. These have non-normal limiting distributions. Such a dichotomy into 'quantile theory' and 'extreme value theory' is helpful. However, there are also intermediate situations where r is a more general function of n."

It is not clear how to deal with this when the value of n is fixed (bear in mind that David's n corresponds to our $2^n - 1$), but it indicates that the m-th most extreme order statistics for some unknown value of m (or some extreme ratio m/n) may also fail to be described accurately by the Normal approximation, not just the single highest and lowest. Another useful topic for future research would be to conduct investigations into the value of this m, or the ratio m/n, for various values of (N and n).

The maximum-bias model.

In this model, we can use the maximum absolute bias of all the related approximations to calculate the data complexity in the same way that the bias of a single linear approximation is used in conventional linear cryptanalysis.

The log-likelihood ratio.

For the reasons stated in Subsection 6.3.1, we will not be covering the use of the log-likelihood ratio in nonlinear cryptanalysis in detail in this thesis. However, most of the results of Hermelin et al. [70] apply both to multidimensional linear and nonlinear cryptanalysis, and so we wish to briefly address an error in their earlier work here.

The following equation is derived using the incorrect approximation $a \approx (b^2/2) - log_2(M+1)$:

$$a \approx (\sqrt{NC} - \Phi^{-1}(P_s))^2/2 - log_2(M+1) \tag{6.3.4}$$

Using the approximation $b \approx \Phi^{-1}(1-2^{-a-log_2(M+1)})$ instead, we obtain a very different equation:

$$a \approx 0.72(\sqrt{NC} - \Phi^{-1}(P_s))^2 + log_2(\sqrt{NC} - \Phi^{-1}(P_s)) + 1.325 - log_2(M+1) \tag{6.3.5}$$

Where appropriate, we have used Mathematica to solve the above equation and recalculate N for the LLR-based multidimensional attacks which we compare our nonlinear attacks to. The issues relating to the linear hull effect are by no means resolved through this correction, and we emphasise that research into the effectiveness of the LLR statistic in cryptanalysing ten or more rounds of a Serpent-like cipher is sorely needed!

6.3.3 When the cipher is not a substitution-permutation network.

For a cipher such as DES, the procedure is not so straightforward to adapt, and it may not be possible to do so in all cases. Let us consider a situation in which we have incorporated nonlinear approximations into Matsui's linear attack on the full DES [180]. Let us start by adapting the part of the attack based on his Equation 4:

We cannot replace the third-round xor of bits [7, 18, 24] with a nonlinear term due to the xors which are applied to it; a nonlinear term in variables z[i] is not equal to the xor of (the same nonlinear term in variables x[i]) with (the same nonlinear term in variables y[i]). Therefore, we cannot incorporate nonlinear components into the first round of the approximation.

The final round is another matter. We can replace the linear approximation to DES S5 with any nonlinear approximation with output bitmask 15 and nonlinear input component. There are several of these such that at least one of the set of relateds has bias ± 24; at present our metaheuristic algorithm has found sixty-two. Of these (numbering the S-box input bits from 1 to 6, with the MSB being 1):

- One approximation uses S-box input bits 1, 2, 5, 6. Although we number the S-box input bits x_i differently, this is the approximation found by Knudsen and Robshaw

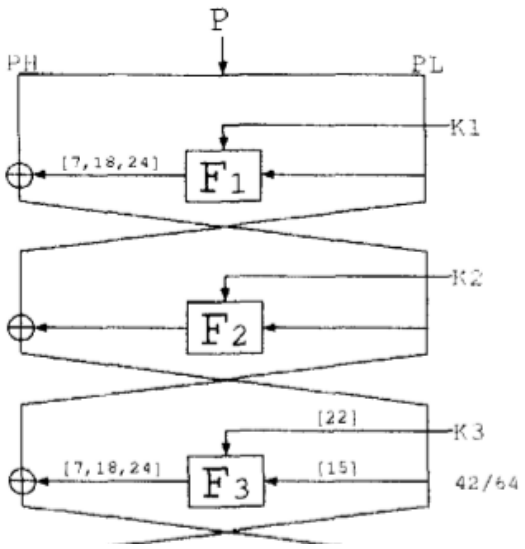

Figure 6.5: Diagram showing the first three rounds of DES in Matsui's attack. The numbers in square brackets indicate the active bits using Matsui's indexing system [180], and the fraction on the right shows the probability p of the first round's linear approximation.

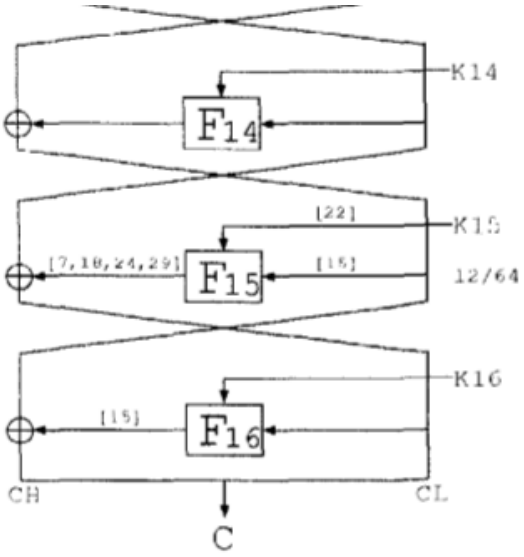

Figure 6.6: Diagram showing the last three rounds of DES in Matsui's attack. As in Figure 6.5, the numbers in square brackets indicate the active bits using Matsui's indexing system [180], and the fraction on the right shows the probability p of the penultimate round's linear approximation.

[164].

- Thirty-one approximations use input bits 1, 2, 4, 5, 6.
- Thirty approximations use input bits 1, 2, 3, 5, 6.

Let us look at how we can adapt the new procedure to the first of these cases.

First of all, we will need to decrypt S-boxes in Round 16 to expose the data bits relevant to S5 in Round 15:

Round 15 data bit	Round 16 data bit
S5 input bit 1	S3 output bit 2
S5 input bit 2	S1 output bit 2
S5 input bit 3	S2 output bit 4
S5 input bit 4	S6 output bit 4
S5 input bit 5	S4 output bit 2
S5 input bit 6	S8 output bit 4

Table 6.4: S-box output bits in Round 16 corresponding to the S5 input bits in Round 15.

In addition, we will need to guess key bits for S5 in round 15. Some of these will already have been guessed:

Round 15 key bit	Round 16 key bit
S5 input bit 1	S6 input bit 2
S5 input bit 2	S6 input bit 3
S5 input bit 3	S6 input bit 1
S5 input bit 4	S8 input bit 4
S5 input bit 5	S8 input bit 1
S5 input bit 6	N/A (main key bit 53, numbering from 0 to 55)

Table 6.5: Key bits corresponding to S5 input bits in Round 15, and their Round 16 counterparts where applicable.

Finally, we still have to guess key bits for S5 in round 1:

Round 1 key bit	Round 16 key bit
S5 input bit 1	S8 input bit 2
S5 input bit 2	Main key bit 52 - S7 bit 6
S5 input bit 3	S8 input bit 6
S5 input bit 4	N/A (main key bit 37)
S5 input bit 5	S6 input bit 6
S5 input bit 6	Main key bit 55 - S7 bit 4

Table 6.6: Key bits corresponding to S5 input bits in Round 1, and their Round 16 counterparts where applicable.

Let us consider the approximation on bits 1, 2, 5 and 6, since this provides the simplest example. We need to guess key bits for four S-boxes (S1, S3, S4, S8) in Round 16; 24 key bits corresponding to 22 text bits. To allow C to have the property that $C_{ij} = f(i \oplus j)$, we will work with the 24 bits resulting from applying the expansion to the text bits. We also need to guess four key bits in Round 1 - we will need to introduce two dummy key bits to correspond to the six input bits of S5 - despite knowing that they should share the same values as key bits 2 and 6 of S8. So far, we have $|k_1| = 30$. We also have four active text bits in the left-hand ciphertext block, which we cannot now simply xor together and treat as part of a larger xor of bits. These require us to introduce four more dummy key bits with the value zero, in addition to the dummy key bit for the xored bits in the left hand of the plaintext and right hand of the ciphertext. We have $|k_1| = 35$. Since one of the four key bits at the input to DES S5 in round 15 is active in round 16, we also have $|k_2| = 3$.

The question of estimating the complexity of a partial encryption/decryption in terms of DES encryptions also arises. Matsui encrypts one S-box, decrypts another, and xors various bits; as the full DES involves (8x16) = 128 S-boxes, we will estimate the complexity of each partial encryption/decryption in Matsui's analysis phase to be $2/128 = 1/64$ of a full DES encryption.

In our case, this is more complicated. We encrypt one S-box and decrypt four, after which we need to compute the following algebraic expression on S5's input bits $2^{|k_2|} = 8$ times:

$$1 \oplus x_5 \oplus x_5x_6 \oplus x_2x_6 \oplus x_1x_5 \oplus x_1x_2 \oplus x_1x_5x_6 \oplus x_1x_2x_6$$

(Note that this was the expression used as an example before.)

We therefore estimate the time required for each partial encryption/decryption to be

$5/128 + 8/128 = 13/128$ of the time required for a full DES encryption.

This specific attack, although it breaks the DES and in spite of its improved bias, turns out to have poorer data complexity than that of Matsui [180]. We will explain later on how Matsui's attack - in effect a combination of two separate attacks - is able despite its lower bias to perform more effectively. For now, though, with the method defined, we have enough information to design cost functions and run experiments, and it is therefore time to discuss the metaheuristic algorithm.

6.4 The use of simulated annealing to evolve nonlinear approximations.

In the previous application of simulated annealing to this problem by Clark et al. [78, 76], each nonlinear approximation was represented as follows:

- A global constant, k, determined the maximum number of S-box input bits that could be involved in the nonlinear component of the approximation. The number n of input bits was 9, and values of k between 2 and 8 were used in experiments.
- The nonlinear equation, on k of the n input bits, was represented by its truth table (an array of 1s and 0s). As stated, this framework did not take into account the related approximations.
- The linear equation on the output bits was represented by an m-bit bitmask (m being 32 in this case), with 1s corresponding to the positions of the bits involved. Most C/C++ compilers could easily accommodate this using an unsigned long integer.
- A "projection" containing the information on which of the input bits were involved in the approximation was represented using an array of size k.

The cost function multiplied the absolute bias of the approximation by -1, and returned the result. The initial acceptance rate was set at 0.6. The move function was somewhat unusual for a simulated annealing algorithm, in that it chose one of four move types at random. Three user-supplied parameters dictated the relative probabilities of changes to the nonlinear component's truth table, the linear component's bitmask, and the projection as follows:

$0 \leq P\_NL\_TT \leq P\_BITMASK \leq P\_SWAP\_USED\_UNUSED \leq 1.0$
$u \leftarrow RAND(0, 1)$
if $u < P\_NL\_TT$ **then**
    A randomly chosen bit in the nonlinear component's truth table is flipped.
else if $P\_NL\_TT \leq u < P\_BITMASK$ **then**
    The linear component's bitmask is changed.
    A new bitmask is chosen uniformly at random from the set of m-bit integers.
    (This causes 2^{m-1} bits in the linear component's truth table to change.)
else if $P\_BITMASK \leq u < P\_SWAP\_USED\_UNUSED$ **then**
    The projection is altered.
    An unused input bit replaces one of those involved in the nonlinear function.
else
    The ordering of the bits in the projection is changed
end if

Clark et al. experimented with $(0.25 \leq P\_NL\_TT \leq 0.45)$, $(0.25 \leq P\_BITMASK \leq 0.45)$, and $P\_SWAP\_USED\_UNUSED \in \{0.5, 1.0\}$.

For changes in the truth table of the nonlinear component, we have reason to believe a smooth search landscape is defined for the move function as described.

Let there be k bits involved in the nonlinear approximation of a single S-box S. Then, if the nonlinear component of the approximation acts on the input bits, there are $(n - k)$ input bits not involved in it (and $(m - k)$ if the nonlinear component acts on the output bits).

The truth table of the nonlinear component of the approximation will contain 2^k entries. Let us consider a "padded" truth table for the approximation, containing the value of the nonlinear expression for every possible value of the bits at the same "end" as the nonlinear component. This truth table will contain 2^{n-k} (or 2^{m-k}) copies of the truth table entry for any choice of the k involved bits. Changing one bit of the nonlinear approximation's truth table will change the values of these copies in the padded truth table, and no other bits.

Now, let us consider the full truth table of the nonlinear approximation, containing the value of the nonlinear expression for every possible value of the S-box's input bits. Clearly if the nonlinear expression is in terms of the input bits, this will be identical to the padded truth table. If not, we compute the full truth table using the equation $FULL\_TRUTH\_TABLE[i] = PADDED\_TRUTH\_TABLE[S(i)]$. For a bijective S-box, changing one bit in the basic truth table of the nonlinear component will change precisely

$2^{m-k} = 2^{n-k}$ bits in this table. For a balanced S-box with more input than output bits (such as a DES S-box), changing one bit in the basic truth table of the nonlinear component will change $2^{n-m} \cdot 2^{m-k} = 2^{n-k}$ bits in the full truth table.

Since we can upper bound the number of changes to the truth table of the approximation's nonlinear component by 2^{n-k}, which can be as low as 2 in the circumstances described if the nonlinear approximation uses $(n-1)$ input bits, and since the truth table of the linear component does not change, none of the 2^k related approximations can change bias by more than 2^{n-k} when a move of this sort is made - and this acts in turn as an upper bound on changes in the absolute values of their biases.

However, for the other possible moves, experiments have shown that a smooth search landscape is not defined:

- A change to the bitmask changes the value of precisely half the bits in the linear component's truth table, meaning that for an S-box mapping $GF(2)^n$ to $GF(2)^m$, the change in an approximation's bias from such a move is upper bounded by 2^{n-1} - an upper bound so high as to be almost meaningless.

  As an example, consider the following nonlinear approximation to DES S5. The nonlinear component of the approximation involves input bits 0, 1, 3, 4, 5 and has full truth table

  11111111111111111000110010001100111111111111111100000010000000010

  This approximation holds with bias 22 when the linear component has bitmask 1111. However, changing the bitmask to 1101 results in its holding with bias 0, and 0 is 12 less than the smallest absolute bias of any of the original related approximations.

- Changing the order of the bits involved in the nonlinear approximation (for example, where approximation input bit x_{a_0} is x_0 and x_{a_1} is x_1, exchanging their positions so that $x_{a_0} = x_1$ and $x_{a_1} = x_0$) will have no effect on the basic truth table of the approximation, but can affect several bits in the full truth table (in this case, as many as 2^{n-1}). In the case of our previous example, this causes the bias to drop from 22 to 2. Moreover, the highest absolute bias of any of the related approximations drops to 4, when the lowest had previously been 12.

- Finally, we consider swapping one of the bits involved in the nonlinear approximation with an uninvolved bit. One of the approximations related to our previous example

held with bias -20. Replacing $x_{a_1} = x_1$ with $x_{a_1} = x_2$ changed its full truth table in 26 places and reduced this bias to 0. The largest absolute bias among any of the related approximations was now 8, 4 lower than the previous minimum.

There is evidence that this affected the behaviour of the search in Clark et al.'s experiments. In their paper, it is stated that for "almost all the executions tried" the search began with an initial period in which there was little improvement in the quality of the best approximation found, lasting approximately 500,000 moves. After this, a period of rapid and almost uninterrupted improvement began, lasting for approximately 500,000 to 700,000 moves, before the level of improvement tailed off.

We believe that the period of improvement began when the temperature of the annealing algorithm had dropped to a point where non-improving moves were very unlikely to be accepted, and that early on in this period, a sequence of moves, all acting on the truth table of the nonlinear component, were accepted. These moves increased the absolute bias of the approximation to the extent that any other sort of move was most unlikely to be accepted due to the unpredictable (but increasingly likely to be negative) effects of such moves on said value. In other words, the algorithm spent this period hill-climbing, with a move function that was limited to changing bits in the truth table of the nonlinear component, before slowing down as it approached a local optimum.

To exploit this, we significantly increase the probability of truth table changes being chosen as the move; we chose to increase probability to 0.9. Since we were attempting to find nonlinear approximations to replace the first and last round components of existing linear approximations to ciphers, the bitmask was assigned a value at the start of the search and remained static thereafter. We still allowed the search to make moves of the other two types (with probability 0.05 each) to see if it would "home in on" particular choices of projection which would result in biases of higher magnitude than others; this was indeed the case.

We focused primarily on S-boxes from ciphers which were

1. such that linear cryptanalysis or a variant/derivative thereof has been used in a significant attack on the cipher or a reduced-round variant thereof.

2. significant, due to being or having been widely used, or being considered a viable alternative to AES, or being a promising new lightweight cipher...

The three ciphers which best satisfied both of these criteria were DES [191], PRESENT [40] and Serpent [5].

Our original experiments utilised various cost functions.

1. Let the number of related approximations be denoted R, and let ϵ_i denote the bias of the ith related approximation ($0 \leq i < R$).

   We initially rewarded high values for the sums-of-squares of the biases; with the cost being:

   $$2^{3n-3} - \sum_{i=0}^{R-1} \epsilon_i^2$$

2. As it became apparent that the related approximations were not always statistically independent of each other; and furthermore that the sample biases in the vector v at the end of the cryptanalysis would not be either, we attempted to refine the first cost function to address this issue. The next cost function was identical to the first, but did not count the biases of related approximations with truth tables that were either identical to the truth tables of previous relateds, or could be obtained from such by flipping all the bits in the truth table. (We refer to a truth table that can be obtained in this way as a "bit-flip" of the previous truth table from which it is obtained.) This was not sufficient to address the issue of statistical dependency among the linear algorithms, and we began to focus our cost functions more on the maximum bias model.

3. We rewarded the highest absolute bias:

   $$cost = \left(2^{n-1} - \max_{(0 \leq i < R)} |\epsilon_i|\right)$$

   In situations where the maximum-bias model is used, and obtaining the k_1 bits is prioritised over obtaining k_2 bits, this strategy makes the most sense. In the attacks on Serpent described below, for instance, the number of k_2 bits compared to the number of k_1 bits was extremely small. This was also the case with the attack on DES, and no cost function used identified a nonlinear approximation such that we could be sure the correct value of k_2 could be identified.

4. We attempted to obtain related approximations such that all biases were high in

magnitude by rewarding high values for the smallest absolute biases:

$$cost = \left(2^{n-1} - \min_{(0 \leq i < R)} |\epsilon_i|\right)$$

This cost function, when tried, merely returned the highest bias linear approximation for the specified bitmask in all cases, suggesting (though this is not a matter of certainty) that nonlinear approximations with all relateds having bias in excess of the best linear approximation may not exist.

5. In an attempt to find cost functions suited for obtaining both k_1 and k_2 bits, we then tried $cost = 2^{2n-2} -$ (max. bias - 2nd highest bias). In the case of the 4×4 S-boxes, this did not find any nonlinear approximations that cost function 3 above had not. In the case of DES S5, which we were targeting due to its presence in the final round of Matsui's linear attack, this found approximations such that the maximum bias among the relateds had magnitude ranging from 18 to 22; and such that the second-highest bias was 12 lower. The bias 24 approximations found by cost function 3 were, however, considered more effective in recovering the k_1 bits due to the extent to which they reduced the data complexity. It should be noted that we considered the recovery of the k_1 bits to be a much higher priority than the recovery of the k_2 bits, since achieving the first objective was necessary to achieve the second.

6.5 Experiments on various ciphers, and application to their cryptanalyses.

6.5.1 AES.

We ran several experiments with the AES S-box and a cost function seeking to reward the maximum absolute bias of all related approximations. Using 600,000 inner loops, 500 outer loops, cooling factor 0.97 and initial acceptance rate 0.95, these yielded several approximations with absolute biases ranging from 64 to 72 (albeit with much lower-bias relateds). Some of these approximations were linear on the S-box's input bits, and some were linear on the output bits - in both cases, several approximations with bias ± 72 were obtained. For all of the input and output bitmasks involved, the bias of the best linear approximation was ± 16.

Since all AES S-box co-ordinate functions are affine-equivalent [127], the question arises as to whether the absolute bias of the best nonlinear approximation to a Boolean function is affine invariant. While more experimentation would be needed to gain evidence for this, if true it would mean that for all input and output bitmasks, the AES S-box would have at least one nonlinear approximation with bias ± 72 (and for some bitmasks, we have already found more than one nonlinear approximation with such a bias.)

While some of the approximations with bias ± 72 were balanced, most were not. All balanced approximations found so far with absolute bias 72 have been linear on the input bits and nonlinear on the output bits, although further experimentation may yield balanced approximations with this absolute bias that are linear on the output bits.

6.5.2 Serpent.

In the literature, various linear, differential-linear, multiple linear and multidimensional linear attacks on reduced-round Serpent are described. These fall into two categories, those based on Collard et al.'s approximations [98, 99, 100, 196] and those based on the approximation of Dunkelman, Keller et al. [27, 31, 123]. In Appendix C, we point out a few errors in the descriptions of Dunkelman et al.'s approximation as found in the literature.

Of the existing attacks on reduced-round Serpent to utilise linear approximations, those described by Collard et al. included a 2R attack on 11 rounds of Serpent, utilising a 9-round approximation, and later using the new analysis phase to improve time complexity.

The same 9-round approximation was later used by Nguyen et al. [196]. In their paper, a 56-dimensional approximation to the preceding round using several input bitmasks was connected to the first round of the original approximation; yielding a 10-round multidimensional approximation which resulted in the best attacks so far against reduced-round Serpent (up to 12 rounds).

However, we believe that the data complexities of these attacks have been underestimated.

Let C denote capacity, and p the probability that the linear approximation holds (so that $(p - 1/2)$ is the bias). In Nguyen et al.'s work [196], the data complexity N specified in each case is $4C^{-1}$. This figure appears to have been chosen to match the values for N used by Collard et al., in which key ranking was not used and values of N equal to $4C^{-1}$ were used in the multiple linear attacks. However, Collard et al. also used N equal to $4 \cdot |p - 1/2|^{-2}$ in the conventional linear attacks (apparently to obtain a success probability

of 0.785 as predicted by Matsui [179]), and $4C^{-1}$ is equal to 1/(sum of squares of biases). To obtain internal consistency, $N = 16C^{-1}$ would have been needed.

Moreover, Matsui's Table 3 [179], containing the success probabilities for various multiples of $|p - 1/2|^{-2}$, is calculated from the double integral in his Lemma 5 based on the assumption that the TPS length l is equal to 6. Selçuk [213] states that there is a tendency to base complexity calculations for linear cryptanalytic attacks on results from Matsui's work that specifically applied to his attacks on DES. He presents an alternative double integral for use in calculating the advantage when key ranking is not utilised.

Since this double integral is different from that in Matsui's Lemma 5, the question of how each was derived, and which is the better choice, arises. Neither is limited to the case of $l = 6$, and different values may easily be input.

Examining Matsui's original integral, we observe first of all a typographical error. The limits of the internal integral include the term $(p - 1/2)$. This is clearly meant to be $|p - 1/2|$, since if not, two attacks using approximations with identical magnitude but different sign of bias would have significantly different complexity.

The intent behind the double integral would be as follows: for each possible value x for the empirical absolute bias when the correct key is used, calculate the probability that the empirical bias y for all other k_i satisfies $(-x < y < x)$. By integrating this over all $x > 0$, the probability of success of the attack is obtained.

$$\int_0^\infty \left(\prod_{k_i \neq k_0} Pr(-x < y < x) \right) f_R(x) \mathrm{d}x$$
$$= \int_0^\infty \left(\prod_{k_i \neq k_0} \int_{-x}^{x} f_{W_{k_i}}(y) \mathrm{d}y \right) f_R(x) \mathrm{d}x$$

However, examining the integrals in more depth makes it clear that a Normal distribution is assumed for x, whereas if x represents absolute bias, it would have a Folded Normal distribution. We therefore restate the intent as follows: for each possible value x for the bias when the correct key is used, calculate the probability that the empirical bias y for all other k_i satisfies $(min(-x, x) < y < max(-x, x))$. By integrating this over $(\infty > x > -\infty)$, the probability of success of the attack is obtained.

$$\int_{-\infty}^{0} \left(\prod_{k_i \neq k_0} Pr(x < y < -x) \right) f_R(x)\mathrm{d}x + \int_{0}^{\infty} \left(\prod_{k_i \neq k_0} Pr(-x < y < x) \right) f_R(x)\mathrm{d}x$$

$$= \int_{-\infty}^{0} \left(\prod_{k_i \neq k_0} \int_{x}^{-x} f_{W_{k_i}}(y)\mathrm{d}y \right) f_R(x)\mathrm{d}x + \int_{0}^{\infty} \left(\prod_{k_i \neq k_0} \int_{-x}^{x} f_{W_{k_i}}(y)\mathrm{d}y \right) f_R(x)\mathrm{d}x$$

$$= \int_{-\infty}^{0} \left(\prod_{k_i \neq k_0} \int_{x}^{-x} \frac{1}{\sigma_W} \phi\left(\frac{y - \mu_W}{\sigma_W} \right) \mathrm{d}y \right) \frac{1}{\sigma_W} \phi\left(\frac{x - \mu_R}{\sigma_R} \right) \mathrm{d}x$$
$$+ \int_{0}^{\infty} \left(\prod_{k_i \neq k_0} \int_{-x}^{x} \frac{1}{\sigma_W} \phi\left(\frac{y - \mu_W}{\sigma_W} \right) \mathrm{d}y \right) \frac{1}{\sigma_R} \phi\left(\frac{x - \mu_R}{\sigma_R} \right) \mathrm{d}x$$

Selçuk simplifies the calculation by assuming that the Wrong-Key Randomization Hypothesis (WKRH) applies. That is, for all incorrect candidate values $k_i \neq k_0$ for the TPS k_1, he assumes that the biases have identical underlying probability distributions with mean 0. We will also do so here:

$$\int_{-\infty}^{0} \left(\int_{x}^{-x} \frac{1}{\sigma_W} \phi\left(\frac{y - \mu_W}{\sigma_W} \right) \mathrm{d}y \right)^{2^l - 1} \frac{1}{\sigma_R} \phi\left(\frac{x - \mu_R}{\sigma_R} \right) \mathrm{d}x$$
$$+ \int_{0}^{\infty} \left(\int_{-x}^{x} \frac{1}{\sigma_W} \phi\left(\frac{y - \mu_W}{\sigma_W} \right) \mathrm{d}y \right)^{2^l - 1} \frac{1}{\sigma_R} \phi\left(\frac{x - \mu_R}{\sigma_R} \right) \mathrm{d}x$$

Let us integrate by substitution. Firstly, let $u = ((x - \mu_R)/\sigma_R)$:

$$
\begin{aligned}
&\int_{-\infty}^{\frac{0-\mu_R}{\sigma_R}} \left(\int_{x}^{-x} \frac{1}{\sigma_W} \phi\left(\frac{y-\mu_W}{\sigma_W} \right) \mathrm{d}y \right)^{2^l-1} \phi(u)\mathrm{d}u \\
&+ \int_{\frac{0-\mu_R}{\sigma_R}}^{\infty} \left(\int_{-x}^{x} \frac{1}{\sigma_W} \phi\left(\frac{y-\mu_W}{\sigma_W} \right) \mathrm{d}y \right)^{2^l-1} \phi(u)\mathrm{d}u \\
&= \int_{-\infty}^{\frac{0-\mu_R}{\sigma_R}} \left(\int_{u\sigma_R+\mu_R}^{-u\sigma_R-\mu_R} \frac{1}{\sigma_W} \phi\left(\frac{y-\mu_W}{\sigma_W} \right) \mathrm{d}y \right)^{2^l-1} \phi(u)\mathrm{d}u \\
&+ \int_{\frac{0-\mu_R}{\sigma_R}}^{\infty} \left(\int_{-u\sigma_R-\mu_R}^{u\sigma_R+\mu_R} \frac{1}{\sigma_W} \phi\left(\frac{y-\mu_W}{\sigma_W} \right) \mathrm{d}y \right)^{2^l-1} \phi(u)\mathrm{d}u
\end{aligned}
$$

Selçuk states that the bias of the correct key has a Normal distribution, with mean $\mu_R = (p-1/2)$ and variance $\sigma_R^2 = 1/4N$. The figure for the variance appears to be derived from Junod [146]; we have no reason to doubt it.

If $p > 1/2$, the mean $(p - 1/2)$ is equal to $|p - 1/2|$. Let $N = a\times|p - 1/2|^{-2}$, and we have:

$$
\frac{0-\mu_R}{\sigma_R} = 2\sqrt{N}(0 - |p-1/2|) = 2\sqrt{a}|p-1/2|^{-1}(0-|p-1/2|) = -2\sqrt{a}
$$

Assuming $a > 2$, $(0-\mu_R)/(\sigma_R) < -2\sqrt{2} \approx -2.828$. $\Phi(-2\sqrt{2}) \approx 1/427$, implying that the contribution of the first integral will be negligible and that we can approximate the whole expression with:

$$
\int_{\frac{0-\mu_R}{\sigma_R}}^{\infty} \left(\int_{-u\sigma_R-\mu_R}^{u\sigma_R+\mu_R} \frac{1}{\sigma_W} \phi\left(\frac{y-\mu_W}{\sigma_W} \right) \mathrm{d}y \right)^{2^l-1} \phi(u)\mathrm{d}u
$$

If $p < 1/2$, the mean $(p - 1/2)$ is equal to $-|p - 1/2|$. Again, let $N = a\times|p - 1/2|^{-2}$, and we have:

$$
\frac{0-\mu_R}{\sigma_R} = 2\sqrt{N}(0 + |p-1/2|) = 2\sqrt{a}|p-1/2|^{-1}(0+|p-1/2|) = 2\sqrt{a}
$$

Still assuming $a > 2$, $(0-\mu_R)/(\sigma_R) > -2\sqrt{2} \approx 2.828$. $P(u > 2\sqrt{2}) = 1 - \Phi(2\sqrt{2}) \approx 1/427$, implying that the contribution from the second integral will be negligible and that we can approximate the whole with:

$$\int_{-\infty}^{\frac{0-\mu_R}{\sigma_R}} \left(\int_{u\sigma_R+\mu_R}^{-u\sigma_R-\mu_R} \frac{1}{\sigma_W} \phi\left(\frac{y-\mu_W}{\sigma_W}\right) \mathrm{d}y \right)^{2^l-1} \phi(u)\mathrm{d}u$$

On the other hand, for $p < 1/2$, we could have defined x as $-1\times$ the bias at the start, and obtained the same approximation as for the case $p > 1/2$. This means that the probability of success can in both cases be approximated by:

$$\int_{\frac{0-\mu_R}{\sigma_R}}^{\infty} \left(\int_{-u\sigma_R-\mu_R}^{u\sigma_R+\mu_R} \frac{1}{\sigma_W} \phi\left(\frac{y-\mu_W}{\sigma_W}\right) \mathrm{d}y \right)^{2^l-1} \phi(u)\mathrm{d}u$$

with $\mu_R = |p - 1/2|$ and $\sigma_R = 1/2\sqrt{N}$. (If this were not the case, we would have been faced with a situation where the sign of the approximation's bias affected the performance of the attack despite the cryptanalyst discarding this information and taking the absolute bias in Attack 2.)

Let us now complete the substitution of $|p - 1/2|$ for μ_R and $1/2\sqrt{N}$ for σ_R in the equation above:

$$\int_{-2\sqrt{N}|p-1/2|}^{\infty} \left(\int_{-u/2\sqrt{N}-|p-1/2|}^{u/2\sqrt{N}+|p-1/2|} \frac{1}{\sigma_W} \phi\left(\frac{y-\mu_W}{\sigma_W}\right) \mathrm{d}y \right)^{2^l-1} \phi(u)\mathrm{d}u$$

Then, we integrate by substitution again. Let $v = (y - \mu_W)/\sigma_W$, and we have:

$$\begin{aligned}
&\int_{-2\sqrt{N}|p-1/2|}^{\infty} \left(\int_{\frac{-u/2\sqrt{N}-|p-1/2|-\mu_W}{\sigma_W}}^{\frac{u/2\sqrt{N}+|p-1/2|-\mu_W}{\sigma_W}} \phi(v)\mathrm{d}v \right)^{2^l-1} \phi(u)\mathrm{d}u \\
&= \int_{-2\sqrt{N}|p-1/2|}^{\infty} \left(\int_{\frac{-u/2\sqrt{N}-|p-1/2|-0}{1/2\sqrt{N}}}^{\frac{u/2\sqrt{N}+|p-1/2|-0}{1/2\sqrt{N}}} \phi(v)\mathrm{d}v \right)^{2^l-1} \phi(u)\mathrm{d}u \\
&= \int_{-2\sqrt{N}|p-1/2|}^{\infty} \left(\int_{-u-2\sqrt{N}|p-1/2|}^{u+2\sqrt{N}|p-1/2|} \phi(v)\mathrm{d}v \right)^{2^l-1} \phi(u)\mathrm{d}u
\end{aligned}$$

- precisely Selçuk's equation. We therefore accept this double integral as correct unless there is reason to believe that the WKRH does not apply, and even then we would use a modified version of Selçuk's equation in preference to Matsui's.

Let us compare the predicted values for the probability of success (denoted P_s) in Matsui's attack on 8-round DES with $l = 6$:

N	$2\|p-1/2\|^{-2}$	$4\|p-1/2\|^{-2}$	$8\|p-1/2\|^{-2}$	$16\|p-1/2\|^{-2}$
P_s ($l=6$, Matsui)	0.486	0.785	0.967	0.999
P_s ($l=6$, Selçuk)	0.589331	0.902745	0.997249	0.999999

Table 6.7: Comparison of success probabilities (calculated numerically using Wolfram Mathematica) for $l=6$ according to Matsui [179] and Selçuk [213]

Clearly, unless Matsui had reason to believe that the wrong-key randomization hypothesis did not hold, his original equation gave pessimistic estimates for the success probability of Algorithm 2 without key ranking.

Tables 6.8, 6.9, and 6.10 give the expected success rates for other values of l, in particular $l=108$, since this is the value of l used in Collard et al.'s maximum advantage attack. We can clearly see from these that $N=4|p-1/2|^{-2}=2^{118}$ was not enough in Collard et al.'s attack on 11-round Serpent to achieve $P_s=0.785$. A reasonably close probability to 0.785 may be achieved with $N=41.5|p-1/2|^{-2}\approx 2^{121.375}$, or an extremely high probability with $N=2^{122}$.

For the same reasons, in Biham et al.'s linear attack on 11-round Serpent, $N=53|p-1/2|^{-2}\approx 2^{121.728}$ is needed instead of $N=4|p-1/2|^{-2}=2^{118}$ to achieve success probability 0.785.

N	$8\|p-1/2\|^{-2}$	$16\|p-1/2\|^{-2}$	$17.6\|p-1/2\|^{-2}$	$32\|p-1/2\|^{-2}$
P_s ($l=44$, Selçuk)	0.028194	0.657866	0.785718	0.999875

Table 6.8: Probabilities of success (calculated numerically) for $a=l=44$.

N	$16\|p-1/2\|^{-2}$	$32\|p-1/2\|^{-2}$	$41.5\|p-1/2\|^{-2}$	$64\|p-1/2\|^{-2}$
P_s ($l=108$, Selçuk)	0.000027	0.229319	0.794093	0.999955

Table 6.9: Probabilities of success (calculated numerically) for $a=l=108$. We were unable to solve for a precise success rate of 0.785.

Furthermore, these are methods used to calculate the data complexity for one-dimensional attacks, and unfortunately the increase in N above would not have been enough to take into account the increase in data requirements resulting from the move from one dimension to fifty-six.

The "Method 2"-based attack on 12-round Serpent [196] aims for 172-bit maximum advantage $a=l$ with $M=(2^{56}-1)$ and capacity $C=2^{-116}$. Based on the discussion above, we assume that the intended probability of success P_s is 0.785. Using this infor-

N	$32\|p-1/2\|^{-2}$	$53\|p-1/2\|^{-2}$	$64\|p-1/2\|^{-2}$
P_s ($l = 140$, Selçuk)	0.007278	0.785316	0.986902

Table 6.10: Probabilities of success (calculated numerically) for $a = l = 140$.

mation to solve Equation 6.3.5 and to compute N from b, we obtain a data complexity of $N \approx 2^{124.39}$.

The "Method 1"-based attack from the same paper is not so easy to estimate data complexity for, since it consists of 2^{128} separate 1R attacks with key guessing on 48 bits in the final round. We can assume that the data complexity for one such 1R attack must lower-bound the value of N in this case, but we believe that it must be an underestimate. Using the same methodology as before, for a capacity of 2^{-114}, we obtain $N \geq\approx 2^{121.275}$.

If differences between the actual and theoretical distributions resulting from the linear hull effect are not too significant, this is still the best attack on reduced-round Serpent to date. However, as the effectiveness of LLR-based nonlinear and multidimensional linear attacks has not to our knowledge been experimentally tested for as many as 12 rounds - or indeed as many as 11 - we are forced to express some doubt as to whether the attack can succeed with the data complexity claimed. This would be a matter for future research.

If we attempt to address this issue by carrying out the attack using the χ^2 statistic instead, then according to the formula given in subsection 6.3.2, if we use the entire codebook of 2^{128} known plaintexts, we obtain an advantage of ≈ 0.279. Since the attack is against 256-bit Serpent, this gives the search phase a time complexity of $\approx 2^{255.721}$. This dominates the complexity of the attack, giving us an approximate overall time complexity of $\approx 2^{255.721}$. Since the success probability is 0.785, and since an exhaustive search of 78.5% of the keyspace would have slightly lower time complexity $\approx 2^{255.651}$, we are not sure that the χ^2 attack could reasonably be viewed as an attack under these circumstances.

The same paper's attacks on 11-round Serpent also underestimate the data complexity. In the case of the attack with twelve active S-boxes in the final round, 48-bit advantage is the implied aim since key ranking is not used. We solve Equation 6.3.5 for $M = (2^{56} - 1)$, capacity 2^{-114} and $P_s = 0.785$, and obtain $N \approx 2^{121.275}$.

In the case of the attack with eleven active final-round S-boxes, we use the same methodology and obtain $N \approx 2^{123.219}$.

(Both of these figures depend on the LLR-statistic remaining usable in spite of the linear hull effect after 11 rounds. If this is not the case, since the same linear characteristic

is used as in the case of the 12-round attacks, we still obtain advantage of only ≈ 0.279 and resultant time complexity $\approx 2^{255.721}$ when using the χ^2 statistic instead - and for the same reasons as before, this probably cannot be considered to constitute an attack.)

If the LLR statistic is used, we assume that the convolution method [136] is used to minimise the complexity of converting the empirical distributions into scores for the various key candidates. The time complexity is still non-negligible compared to the remainder of the attack, being equal to $2^k((6m+13)\cdot 2^m)$ MAs + $2^k((6m+4)\cdot 2^m)$ AOs.

As we see from tables 6.11, 6.12 and 6.13 below, the best *existing* attacks on eleven-round Serpent in terms of data and memory complexity are those of Nguyen et al. Time complexity depends on which of the Serpent key lengths is in use; for the 192 and 256-bit keys, Collard et al. have less key bits remaining to search for and achieve the best time complexity of the existing methods; for the 128-bit key length the faster analysis phase, and reduced time required to encrypt the known plaintexts, of Nguyen et al.'s method dominates the time complexity and makes it the superior attack. The complexity of the search phase gives the nonlinear attack in this thesis the best overall time complexity for the case of 11-round Serpent with 256-bit keys, and it may also be seen that nonlinear cryptanalysis achieves better data complexity than any other known-plaintext - or indeed chosen plaintext - attack on 11-round Serpent with 192 or 256-bit keys.

Using nonlinear attacks to reduce the data complexity of attacking 11-round Serpent-192 and Serpent-256.

It is not clear how, if it is possible at all, to combine nonlinear and multidimensional linear approximations. We therefore focus on modifying Collard et al.'s "Approximation D2", and focus on the version with 12 active S-boxes in the final round.)

The simplest change possible is to replace the (input bitmask 12, output bitmask 10) bias 4 approximation in the first round (affecting bits 16, 17, 18, 19) with the following approximation:

$$x_2 \oplus x_1 \oplus x_1 x_4 = y_1 \oplus y_3$$

The primary approximation has bias 6, and after we eliminate related approximations which are bit-flips of others, we obtain sum-of-squares-of-biases 40. Fortunately, the related approximations are uncorrelated and we obtain the full corresponding increase in capacity should we choose to use the χ^2 model.

20 instead of 15 S-boxes are now activated in the plaintext, increasing the number of

Rounds	Type of attack	Data	Time (analysis)
11	Linear [27]	$2^{121.728}$ KP	$2^{188.1}$ E
11	Linear [27]	$2^{121.728}$ KP	2^{96} PE + 2^{44} PD + $2^{149.73}$ AO + $2^{149.76}$ MA
11	Linear [100]	$2^{121.375}$ KP	2^{60} PE + 2^{48} PD + $2^{117.36}$ AO + $2^{117.4}$ MA
11	Multidim. linear [196]	$2^{121.275}$ KP	2^{48} PD + $2^{113.916}$ AO + $2^{113.943}$ MA
11	Multidim. linear [196]	$2^{123.219}$ KP	2^{44} PD + $2^{109.884}$ AO + $2^{109.91}$ MA
11	Differential-linear [123]	$2^{121.8}$ CP	$2^{135.7}$ MA
11	Nonlinear (this thesis)	$2^{120.467}$ KP	2^{80} PE + 2^{48} PD + $2^{139.6}$ AO + $2^{139.63}$ MA
11	Nonlinear (this thesis)	$2^{117.401}$ KP	2^{60} PE + 2^{76} PD + $2^{149.69}$ AO + $2^{149.72}$ MA
11	Nonlinear (this thesis)	$2^{115.44}$ KP	2^{60} PE + 2^{80} PD + $2^{153.73}$ AO + $2^{153.76}$ MA
11	Differential-linear [123]	$2^{113.7}$ CC	$2^{137.7}$ MA
12	Differential-linear [123]	$2^{123.5}$ CP	$2^{249.4}$ E
12	Multidim. linear (Method 2) [196]	$2^{124.39}$ KP	2^{128} PE + 2^{44} PD + $2^{238.744}$ AO + $2^{238.769}$ MA
12	Multidim. linear (Method 1) [196]	$\geq 2^{121.275}$ KP	2^{128} PE + 2^{48} PD + $2^{241.916}$ AO + $2^{241.943}$ MA

Table 6.11: Attack complexities. In most cases $P_s = 0.785$ (or slightly higher.) The chosen plaintext attacks of Biham et al. have $P_s = 0.84$, and the chosen-ciphertext attack has $P_s = 0.93$. The time complexity for Biham et al.'s linear cryptanalysis varies depending on whether the new analysis method of Collard et al. is used, or whether an earlier analysis method [27] is. E = full encryptions of the reduced round cipher. PE = partial encryptions. PD = partial decryptions. AO = arithmetic operations. KP = known plaintexts. CP = chosen plaintexts. CC = chosen ciphertexts.

Rounds	Type of attack	Time (analysis) summary	Mem	Bits recovered
11	Linear [27]	$2^{188.1}$ E	*	140
11	Linear [27]	$2^{137.08}$ E + $2^{149.76}$ MA	$2^{144.087}$	140
11	Linear [100]	$2^{104.71}$ E + $2^{117.4}$ MA	$2^{112.087}$	108
11	Multidim. linear [196]	$2^{101.266}$ E + $2^{113.943}$ MA	2^{108}	48
11	Multidim. linear [196]	$2^{97.234}$ E + $2^{109.91}$ MA	2^{104}	44
11	Differential-linear [123]	$2^{135.7}$ MA	2^{76}	48
11	Nonlinear (this thesis)	$2^{126.95}$ E + $2^{139.63}$ MA	$2^{134.087}$	128 k_1, 2 k_2
11	Nonlinear (this thesis)	$2^{137.04}$ E + $2^{149.72}$ MA	$2^{144.087}$	136 k_1, 4 k_2
11	Nonlinear (this thesis)	$2^{141.08}$ E + $2^{153.76}$ MA	$2^{148.087}$	140 k_1, 4 k_2
11	Differential-linear [123]	$2^{137.7}$ MA	2^{99}	60
12	Differential-linear [123]	$2^{249.4}$ E	$2^{128.5}$	160
12	Multidim. linear [196]	$2^{225.964}$ E + $2^{238.769}$ MA	2^{232}	172
12	Multidim. linear [196]	$2^{229.136}$ E + $2^{241.943}$ MA	2^{108}	176

Table 6.12: Attack complexities cont. All memory complexities are measured in bytes. The time and memory complexities for Biham et al.'s linear cryptanalysis vary depending on whether the new analysis method of Collard et al. is used, or whether an earlier analysis method [27] is. In the latter case, the relevant sources [27, 100] disagree as to the memory complexity. Based on the bitsliced Serpent implementation and Osvik's new implementation of S6 [5, 202] we estimate $2^{12.65}$ AOs are needed for an 11-round Serpent encryption, ignoring the key schedule as this is only done once, and $2^{12.78}$ AOs for 12-round Serpent. E = full encryptions of the reduced round cipher. PE = partial encryptions. PD = partial decryptions. KP = known plaintexts. CP = chosen plaintexts. CC = chosen ciphertexts.

		Bits remaining		
Rounds	Type of attack	(128-bit key)	(192-bit key)	(256-bit key)
11	Linear [27]	N/A	52	116
11	Linear [27]	N/A	52	116
11	Linear [100]	20	84	148
11	Multidim. linear [196]	80	144	208
11	Multidim. linear [196]	84	148	212
11	Differential-linear [123]	80	144	208
11	Nonlinear (this thesis)	N/A	62	126
11	Nonlinear (this thesis)	N/A	52	116
11	Nonlinear (this thesis)	N/A	48	112
11	Differential-linear [123]	68	132	196
12	Differential-linear [123]	N/A	32	66
12	Multidim. linear [196]	N/A	20	84
12	Multidim. linear [196]	N/A	16	80

Table 6.13: Complexities for attack when $P_s = 0.785$ (or slightly higher) cont.

k_1 key bits attacked to 128. The memory requirements are increased to $2^{(128+2)+4.087} = 2^{134.087}$ bytes, due both to the extra k_1 bits and the four related approximations. The time complexity of the analysis phase also increases, and is dominated by the $4{\cdot}(6{\times}128 + 8){\cdot}2^{128} = 2^{139.6}$ arithmetic operations and $4{\cdot}(6{\times}128 + 26){\cdot}2^{128} = 2^{139.63}$ memory accesses (The multiplication by 4 results from there being four "relateds").

Despite the aforementioned difficulty in comparing memory access complexity to complexity in terms of encryptions, we are able to calculate an estimate for the number of arithmetic operations per reduced-round encryption, by counting the number of operations involved in the optimised "bitslice" implementation of Serpent [5]. In particular, this implementation does not use lookup tables for the S-boxes, but instead uses arithmetic operations to calculate the output values extremely quickly.

If we obtain an optimistic estimate for the number of arithmetic operations per reduced-round Serpent encryption, dividing the attack's AO complexity by this figure will give us a conservative estimate for its time complexity. For this reason, we base our estimate on a version of Serpent in which Osvik's implementation of S6 [202] has replaced the original implementation, allowing one less AO per calculation of S6, and assume that the performance gains of the bitslice implementation are not compromised by this. If future research should provide evidence that this is not in fact possible, we can easily base new estimates on the original version.

(Note that operations such as bitwise xor, which might more often be described as logical operations, are included under the banner of "arithmetic operations" in this case.)

This gives us $2^{12.65}$ AOs per 11-round Serpent encryption, and $2^{12.78}$ per 12-round encryption. Dividing the appropriate figure by $2^{12.65}$, we obtain time complexity of $2^{126.95}$ encryptions + $2^{139.63}$ MAs.

In the χ^2 model, the capacity of the new approximation is equal to 2.5× what it was before, however the increased number of degrees of freedom (4 instead of 1) means that we cannot reduce the data requirements a full 2.5-fold. The number of degrees of freedom is too low for us to use Equation 6.3.3, which in any case heavily underestimates the advantage of the original attack; however if it can be taken as a guide, it indicates that we achieve the same advantage with $2^{120.875}$ known plaintexts instead of $2^{121.375}$.

If we use the maximum-bias approach instead, the capacity is multiplied by $(6/4)^2 = 2.25$. However, we cannot decrease the known-plaintext requirements 2.25-fold, since the increased number of k_1 bits, and the need to deal with 2^2 relateds per outer key guess, effectively raises l to 130. To obtain success probability close to the 0.794 of the original attack, a higher value of $N|p - 1/2|^{-2}$ is needed. 49.75 instead of the previous 41.5 gives us success probability 0.8, and means that N is in fact reduced by a factor of 1.877, to $2^{120.467}$. This is clearly a better option than using the χ^2 statistic.

This approximation involves three k_2 bits. Due to the bit-flipped relateds, we can only recover two of these; the bits corresponding to x_1 and x_4.

There is another bias 4 linear approximation in the first round, and several approximations in the final round with bias ±4, that can be replaced with nonlinear approximations possessing similar properties to the one above. Let us consider a situation in which:

- the entire first round approximation remains linear,
- we replace the final-round S-box approximation $x_3 \oplus x_4 = y_4$ (bias 4; affecting state bits 76 to 79) with $x_3 \oplus x_4 = y_4 \oplus y_3 \oplus y_1 y_3$. This approximation has bias 6, and a statistically independent related with absolute bias 2. All other relateds either have zero bias or are bit-flips of these, so we have sum of squares of biases 40.

  This increases the number of active final-round S-boxes from 12 to 17.
- we also replace one of the final-round $x_1 \oplus x_3 \oplus x_4 = y_2$ approximations (bias 2; the one affecting state bits 96 to 99) with $x_1 \oplus x_3 \oplus x_4 = y_2 \oplus y_1 \oplus y_2 y_4$. The number of active final-round S-boxes increases again, from 17 to 19.

The total number of active S-boxes increases from 27 to 34.

We have replaced a bias 4 ($\text{bias}^2 = 16$) approximation and a bias 2 ($\text{bias}^2 = 4$) approximation with two nonlinear approximations, each being such that the primary approximation has bias 6, and such that the sum of squares of statistically independent biases is equal to 40.

Let us first consider the χ^2 model. In this model, the capacity is multiplied by $(2.5 \times 10) = 25$. Evidence from experiments on an SPN-based cipher in which final-round linear approximations with bias ± 4 were replaced with nonlinear approximations with identical properties to the ones above suggests that a 6.25-fold increase in capacity, mitigated by an increase in the number of degrees of freedom from 1 to 16, results in a reduction in data complexity by a factor of approximately 2^1. This would lead to an estimated $2^{120.375}$ known-plaintext requirement. Since we have a further 2^2-fold increase in capacity on top of this, we estimate that $2^{118.375}$ known plaintexts are required, and that the data requirements for the same advantage as Collard et al.'s original attack are very unlikely to be $\geq 2^{119.375}$. However, these experiments used a smaller value of l, and due to the low number of degrees of freedom, it is not clear how much confidence we can place in these figures.

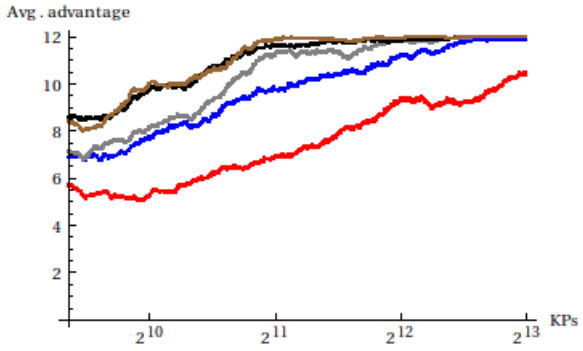

Figure 6.7: Graph showing mean advantages for attack on four round SPN with 4×4 S-boxes using:
linear approximation (red),
nonlinear approximation (Two final-round S-boxes are approximated with "6, 2, bit-flips" approximations of the type used in this section) in χ^2 model (blue),
same nonlinear approximation in maximum-bias model (grey),
multiple nonlinear in χ^2 model with two sets of approximations of this type (black),
multiple nonlinear with same two approximations in maximum-bias model (brown).

If, by contrast, we utilise the maximum-bias model, we replace one bias 4 approxima-

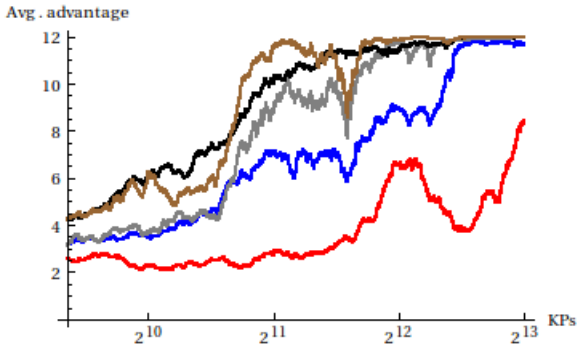

Figure 6.8: Graph showing alternate calculation for average advantage in which the mean rank obtained was input to the formula for advantage.

tion and one bias 2 approximation with two bias 6 approximations, multiplying capacity by $\left(\frac{6\times6}{4\times2}\right)^2 = 20.25$. Since the value of l is effectively increased to 140, this does not simply reduce the known-plaintext requirements to $2^{117.035}$, since we have to increase $N|p-1/2|^2$ to compensate. $N = 53.5|p-1/2|^{-2}$ gives success probability 0.8, and $N = 2^{117.401}$. The memory requirements are increased to $2^{144.087}$. The time complexity of the analysis phase is dominated by $16{\cdot}(6{\cdot}136+26){\cdot}2^{136} = 2^{149.72}$ MAs and $16{\cdot}(6{\cdot}136+8){\cdot}2^{136} = 2^{149.69}$ AOs $\approx 2^{137.04}$ 11-round encryptions.

To reduce the number of known plaintexts further, we could replace another of the $x_1 \oplus x_3 \oplus x_4 = y_2$ approximations with a nonlinear approximation instead of replacing $x_3 \oplus x_4 = y_4$. If we choose the approximation affecting state bits 116-119, we can do this with a total of 35 S-boxes activated, and we obtain time complexity $16{\cdot}(6{\cdot}140+26){\cdot}2^{140} = 2^{153.76}$ MAs and $16{\cdot}(6{\cdot}140+8){\cdot}2^{140} = 2^{153.73}$ AOs $\approx 2^{141.08}$ eleven-round encryptions with memory complexity $2^{148.087}$. Estimated data complexity in the χ^2 model is $2^{116.375}$, but for the reasons given above we view complexity calculations as more reliable in the maximum-bias model. l is in effect increased to 144, resulting in N having to equal $55|p-1/2|^{-2}$ to obtain success probability 0.8 with $N = 2^{115.44}$.

Improving the capacity of the highest-bias approximation of nine rounds of Serpent.

The description of Collard et al.'s approximations [99] includes one of several nine-round linear approximations discovered with bias 2^{-50} (capacity 2^{-98}); the highest bias achieved for a linear approximation of that many rounds. However, none of these approximations

are used in attacks, since the high number of active S-boxes in each would result in attacks with far higher complexity than the then-current state of the art.

Our algorithm found several higher-capacity replacements for linear S-box approximations in the outer rounds (both of which used Serpent S3). These included various approximations in which the various "relateds" were all either:

1. statistically independent, or
2. bit-flips of other relateds, which could safely be ignored

allowing us to calculate the new capacity precisely:

- The first round - nonlinear components in the input bits.
  - We can replace the bias -4 (capacity 64) linear approximation $x_1 \oplus x_4 = y_2 \oplus y_4$ (input bitmask 9, output bitmask 5) with one of the following nonlinear approximations:
    1. $x_2 \oplus x_1x_4 \oplus x_1x_2 = y_2 \oplus y_4$ and relateds. (Primary approximation has bias +6, we can choose a statistically-independent related with bias either 2 or -2, other relateds either have bias 0 or are bit-flips of these.)
    2. $x_4 \oplus x_1 \oplus x_2x_4 = y_2 \oplus y_4$ and relateds. (Primary approximation has bias -6. Again, we can choose a statistically-independent related with bias either 2 or -2, and the other relateds either have bias 0 or are bit-flips of these.)
    3. Other nonlinear approximations such that one related has bias ± 6 exist, but the truth tables of the related approximations are not statistically independent, so we are unable to calculate their capacity when working in the χ^2 model. In experiments on a toy cipher, these appear to have approximately the same capacity, but since optimisations to omit bit-flips and zero-bias relateds cannot be made, they are also much slower to work with.

  Table 6.14 summarises the above:

Nonlinear component	Bias	(2, 4) wrong	(1) wrong	(1, 2, 4) wrong
$x_2 \oplus x_1x_4 \oplus x_1x_2$	+6	-6	-2	+2
$x_4 \oplus x_1 \oplus x_2x_4$	-6	+2	+6	-2

Table 6.14: Nonlinear approximations to S3 with output bitmask 0101.

– We can also replace the bias 4 (capacity 64) linear approximation $x_2 \oplus x_3 = y_1 \oplus y_2$ (input bitmask 6, output bitmask 12) with one of various nonlinear approximations with very similar properties to those found in the above case:

1. $x_2 \oplus x_3x_4 = y_1 \oplus y_2$ and relateds. (Primary approximation has bias +6, we choose an independent related with bias either 2 or -2, all other relateds have either bias 0 or are bit-flips of the preceding.)
2. $x_3 \oplus x_2x_4 = y_1 \oplus y_2$ and relateds. (Primary approximation has bias +6, again we choose an independent related with bias either 2 or -2, and all others have zero bias or are bit-flips of the preceding two.)

As before, other nonlinear approximations with bias ±6 primary approximations but statistically dependent relateds also exist.

Nonlinear component	Bias	(2) wrong	(3) wrong	(2, 3) wrong
$x_2 \oplus x_3x_4$	+6	-6	-2	+2
$x_3 \oplus x_2x_4$	+6	-2	-6	+2

Table 6.15: Nonlinear approximations to S3 with output bitmask 1100.

- The final round - nonlinear components in the output bits.

  – We can replace the bias 4 linear approximation $x_3 \oplus x_4 = y_4$ with one of two nonlinear approximations with similar properties to those presented above. These are: $x_3 \oplus x_4 = y_4 \oplus y_3 \oplus y_1y_3$ and $x_3 \oplus x_4 = y_4 \oplus y_3 \oplus y_3y_4 \oplus y_1y_4 \oplus y_1y_3$.

Nonlinear component	Bias	(4) wrong	(1, 3, 4) wrong	(1, 3) wrong
$y_4 \oplus y_3 \oplus y_1y_3$	+6	-6	-2	+2
$y_4 \oplus y_3 \oplus y_3y_4 \oplus y_1y_4 \oplus y_1y_3$	+6	-2	-6	+2

Table 6.16: Nonlinear approximations to S3 with input bitmask 0011.

This pattern occurs fairly frequently.

– The bias 4 approximation $x_1 \oplus x_3 \oplus x_4 = y_1$ (which occurs three times) can be replaced with one of the *four* approximations in Tables 6.17 and 6.18:

– The bias -4 approximation $x_1 \oplus x_2 \oplus x_3 = y_3$ can be replaced with either of the two approximations in Table 6.19 (both capacity 160):

Nonlinear component	Bias	(1) wrong	(1, 2) wrong	(2) wrong
$y_2 \oplus y_1 \oplus y_2y_4$	+6	-6	-2	+2
$y_2 \oplus y_2y_4 \oplus y_1y_4$	+6	-2	-6	+2

Table 6.17: First set of nonlinear approximations to S3 with input bitmask 1011.

Nonlinear component	Bias	(1) wrong	(1, 3) wrong	(3) wrong
$y_4 \oplus y_3 \oplus y_1 \oplus y_3y_4$	-6	+6	+2	-2
$y_3 \oplus y_3y_4 \oplus y_1y_4$	+6	-2	-6	+2

Table 6.18: Second set of nonlinear approximations to S3 with input bitmask 1011.

Nonlinear component	Bias	(1, 3, 4) wrong	(1, 3) wrong	(4) wrong
$y_4 \oplus y_1 \oplus y_3y_4 \oplus y_1y_4 \oplus y_1y_3$	+6	-6	-2	+2
$y_4 \oplus y_1 \oplus y_3y_4 \oplus y_1y_4$	+6	-2	-6	+2

Table 6.19: Nonlinear approximations to S3 with input bitmask 1110.

Although we are likely to encounter the same issues with increased TPS size and time complexity of handling the relateds as before, in the maximum-bias model this gives us several nonlinear approximations to nine-round Serpent with bias $\pm 2^{-45.9}$ instead of 2^{-50}. In the χ^2 model, we can replace the highest-capacity approximation to 9-round Serpent known so far (capacity $4 * (2^{-50})^2 = 2^{-98}$) with several different approximations with capacity $\approx 2^{-88.75}$.

This is unlikely to be of use in practice - the original nine-round linear approximation had too many active S-boxes in the plaintext and ciphertext to be used in a feasible attack, and this approximation only exacerbates the same problem. We include it here merely to demonstrate the potential nonlinear approximations to cipher rounds have to increase bias and capacity.

6.5.3 DES.

# bits in NL function	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	8	8	8	6	8	10	6	8	8	8	6	10	12	18
3	10	10	12	8	14	10	14	10	10	12	12	10	12	16	24
4	14	14	14	14	14	14	20	16	14	14	18	14	18	18	24
5	14	14	18	16	16	18	22	16	16	16	20	16	22	22	28
Best linear:	14	12	8	10	10	12	12	14	8	12	12	12	10	12	18

Table 6.20: DES S1. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds.

# bits in NL function	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	6	8	4	8	8	10	4	6	8	16	8	12	10	14
3	10	10	12	14	12	10	12	8	10	10	16	10	16	14	22
4	14	12	14	16	14	12	20	10	12	14	18	16	20	18	22
5	16	14	16	18	20	16	24	14	18	18	22	20	22	22	24
Best linear:	10	12	10	14	10	8	10	14	12	10	16	10	12	10	12

Table 6.21: DES S2. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds.

# bits in NL function	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	8	8	4	10	8	8	4	8	8	12	8	14	10	16
3	8	10	12	8	14	12	14	6	10	12	16	12	18	14	18
4	12	12	16	12	16	16	16	10	14	16	18	14	18	20	20
5	14	16	18	14	22	22	20	14	18	20	22	20	24	22	22
Best linear:	14	10	12	12	10	12	12	14	12	10	12	10	14	10	16

Table 6.22: DES S3. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds.

# bits in NL function	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	4	8	4	8	8	12	4	8	8	10	8	10	12	16
3	10	8	8	8	12	8	16	10	8	12	16	8	16	16	16
4	12	12	16	12	12	16	18	12	16	12	18	16	18	18	24
5	16	16	20	16	16	20	22	16	20	16	22	20	22	22	32
Best linear:	10	10	12	10	12	16	10	10	16	12	10	12	10	10	16

Table 6.23: DES S4. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds. Note in particular that, for bitmask 15, maximum bias of 32 was achieved - the xor of the four output bits of DES S4 is independent of the sixth input bit.

# bits in NL function	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	6	6	6	4	10	6	10	6	6	10	12	8	14	16	20
3	6	12	8	6	12	10	16	8	10	10	14	10	14	16	20
4	10	12	14	10	14	14	18	12	14	14	18	14	18	18	24
5	16	18	16	12	16	16	20	14	16	18	20	18	22	22	24
Best linear:	10	12	10	14	10	8	10	12	10	12	12	10	14	16	20

Table 6.24: DES S5. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds. Note that for bitmask 4 the best linear approximation has higher magnitude of bias than any of our nonlinears. The linear function on the input bits involves all six x_i, whereas our nonlinear approximations were limited to five to reflect the fact that if all six x_i were exposed to the cryptanalyst, there would be no need to use an approximation.

# bits in NL function	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	6	6	4	6	8	14	6	8	8	14	8	12	12	12
3	12	8	10	8	10	12	18	6	12	10	14	12	14	18	14
4	12	10	12	10	14	14	20	12	16	16	16	16	18	18	18
5	16	14	20	16	16	16	24	14	18	20	22	22	22	22	20
Best linear:	12	12	10	12	10	10	14	12	8	10	14	12	12	12	12

Table 6.25: DES S6. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds.

# bits in NL function	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	8	10	8	10	6	10	6	8	10	14	10	14	16	14
3	6	8	12	12	14	10	12	8	12	12	16	10	14	16	16
4	12	12	16	14	16	14	18	12	16	14	20	14	18	20	20
5	14	18	20	18	20	20	20	14	18	18	22	20	22	22	24
Best linear:	14	10	10	18	10	10	12	12	8	12	14	12	14	16	14

Table 6.26: DES S7. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds.

# bits in NL function	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	6	8	4	8	6	8	6	12	8	12	6	12	16	16
3	8	10	12	12	10	10	16	10	14	12	14	10	16	16	24
4	14	12	14	12	12	14	18	10	14	16	18	16	18	22	28
5	16	18	18	20	18	16	20	16	18	18	20	18	22	24	28
Best linear:	10	12	12	12	10	10	14	10	12	10	12	10	12	16	16

Table 6.27: DES S8. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds.

The approximation on bits 1, 2, 5, 6

As stated, since one of the four key bits at the input to DES S5 in round 15 is active in round 16, we have $|k_2| = 3$. The main approximation has bias 24, and the relateds corresponding to wrong key guesses for the three undetermined bits have bias 16 (in three cases) and 12 (four cases).

- If the χ^2-statistic is used, then the number M of degrees of freedom for the new attack is equal to $2^{|k_2|}$, which for this attack is 8. We cannot re-guess the key bit that was active in Round 16 to take advantage of the biases of the incorrect relateds, since each of these has a truth table obtained by flipping all the bits in the truth table of one of the relateds for the correct value of this key bit; and hence they provide us with no additional information.
- The complexity of the analysis stage would be dominated by:
  - 2^6 PEs, each with complexity 1/128 that of a full encryption.

– 2^{24} PDs, each with complexity 12/128 that of a full encryption. (The total complexity of the PEs and PDs so far equates to $2^{20.585}$ DES encryptions.)

– $(6 \times 35 + 8) \times 2^{35} = 2^{42.77}$ arithmetic operations.

– $(6 \times 35 + 26) \times 2^{35} = 2^{42.88}$ memory accesses.

Equating complexity in terms of arithmetic operations to an estimated complexity in terms of DES encryptions is much more difficult than in the case of Serpent. If we treat the number of gate operations as equivalent to the number of AOs, Kwan's best figures for bitsliced DES [167] give us a total of 6528 AOs for the S-boxes. Biham [24] claims that we need not treat the DES expansion and permutation as requiring any operations in a bitsliced implementation, that the key xor requires 48 operations per round, and the xor of the round function outputs with the left block requires 32. $(32 + 48) \times 16 = 1280$. Reference is also made to 160 CPU load/store instructions per round; due to the small amount of data involved it may be possible to keep these in cache memory, but they clearly complicate the issue. Kwan also notes [168] improved bitslice S-boxes by the developers of the "John the Ripper" password cracking software, which depending on the CPU architecture may be able to use as few as 4208 AOs instead of 6528.

- An additional 2^{35} time would then be required to go through the set of results and eliminate all values corresponding to incorrect values of the dummy key bits. If we count this as part of the analysis phase, its complexity is expected to be negligible compared to the above.

- Since seven of the bits of k_1 were dummies, there would be 28 key bits remaining to handle during the search phase. There are also 28 non-dummy bits in k_1. If we seek to obtain the same advantage as Matsui's linear attack ($a = 13$), then we would need to use key-ranking with the $X = 2^{28-13} = 2^{15}$ highest-scoring keys, and the search phase would have complexity $O(28{\cdot}2^{28})$ to sort the results, plus $(2^{15}{\cdot}2^{28}) \approx 2^{43}$ DES encryptions.

The complexity of the distillation phase is dependent on the change in data complexity. The various related approximations involved are all statistically dependent, with pairwise correlation coefficients of 0.5, and we do not currently have a statistical model or empirical evidence for the effect this would have on the capacity when using the χ^2-statistic. We

therefore use the maximum-bias model, noting that we may not have sufficient data to deduce bits of k_2 due to the high bias of the relateds involved. Although it is not clear precisely how the time complexity so far compares with Matsui's original attack, we will now see that despite the improved bias of the approximation, the data complexity has *worsened*!

It seems as though the data complexity should be reduced by a factor of $24^2/20^2 = 1.44 = 2^{0.526}$, giving overall data complexity, and time complexity for this phase, of $2^{43-0.526} = 2^{42.474}$. Unfortunately, this is not the case. Matsui's attack does in fact consist of two separate attacks with $a = 6.5$, combined to produce one attack with $a = 13$. This low advantage allows Matsui to use less data than would be the case for a direct advantage 13. We can obtain $P_s = 0.85$ for $a = 6.5, N = 2^{42.5}$, but not with $a = 13$ [213]. The reduced advantage would massively increase the attack's time complexity; still breaking DES but much more slowly than Matsui.

The approximations on bits 1, 2, 4, 5, 6

To use these approximations, it is necessary to guess thirty key bits in the final round (for (S1, S3, S4, S6, S8)). The six active key bits in Round 1 now include three guessed bits and three dummy bits, as one of them is input to S6 in round 16. We need five instead of four dummy key bits for the left-hand ciphertext block now, raising the value of $|k_1|$ to 41. The complexity of a partial encryption/decryption is now 6/128 that of a full DES encryption, *plus* the time required to compute the truth tables of the nonlinear functions on bits 1, 2, 4, 5 and 6. $|k_2|$ is reduced to 1, as only one of the five active key bits at the input to S5 in Round 15 is now not active in Round 16. However, we can increase it by re-guessing Round 16 key bits. The minimum possible complexity of a partial encryption/decryption is, therefore, equal to $((6 + 2^1)/128) = (1/16)$ of a full DES encryption

The complexity of the analysis phase is, as a result of this, now at least

- 2^{30} PDs, each with complexity 7/128 that of a full encryption.
- 2^6 PEs, each with complexity 1/128 that of a full encryption. (The total complexity of the PEs and PDs so far equates to $2^{25.8}$ DES encryptions.)
- $(6 \times 41 + 8) \times 2^{41} = 2^{49}$ arithmetic operations.
- $(6 \times 41 + 26) \times 2^{41} = 2^{49.09}$ memory accesses.

In terms of time complexity, this attack is clearly inferior to its predecessor. As for data complexity, the maximum-bias model only equals the attack described above, and the level of statistical dependence among the related approximations means that we do not currently know what its data complexity in the χ^2 model would be.

6.5.4 PRESENT.

Currently, the largest number of rounds of PRESENT attacked is 26 [66, 67]. This (slightly controversial [137]) attack utilises a new form of multidimensional linear cryptanalysis that relies heavily on the existence of multiple linear paths with the same bias.

We note that this paper's formula for the data complexity is almost identical to Hermelin et al.'s Theorem 1 [70], except that it incorporates the assumption that $a \approx b^2$, and replaces the value $4M$ with $8M$.

We have already criticised the original formula and the $a \approx b^2$ assumption; however the replacement of $4M$ with $8M$ in the (denominator) of 6.3.1, resulting in

$$a \approx \frac{(NC(p) - 4\Phi^{-2}(2P_S - 1))^2}{8M} \tag{6.5.1}$$

is a new development, which the author does not explain. Although this is referred to elsewhere [137] as a typographical error; in emails the author has stated that he believes the $8M$ version to be correct, and has introduced the change to bring the theoretical formula more closely into line with empirical evidence for the behaviour of the attack.

Unfortunately, for the 1-bit input/output bitmasks of the S-boxes in the outer rounds of the approximation, we have not been able to find any nonlinear approximations with sufficient capacity and lack of dependence among the related approximations to improve on this attack, nor on its (less controversial) predecessor [201]. Nonlinear approximations for the PRESENT S-box with higher magnitude of bias than linear approximations do exist, but not for the bitmasks required to attach them to the linear approximation involved in this attack.

	Bitmask for linear function of input bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Max bias (NL approx.)	4	4	6	4	4	4	6	4	6	4	8	4	6	4	6
Best linear approx.	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

Table 6.28: PRESENT S-box. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds (3 bits in nonlinear component).

	Bitmask for linear function of output bits.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Max bias (NL approx.)	4	4	6	4	6	4	4	4	6	4	6	4	6	6	6
Best linear approx.	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

Table 6.29: PRESENT S-box. Maximum (absolute) bias found for cost function rewarding maximum bias among relateds (3 bits in nonlinear component).

6.6 Conclusions, and directions for future research.

In this chapter, we have evolved nonlinear approximations for block cipher S-boxes with higher absolute bias than the best-known linear approximations for said boxes. Prior to doing this, we have designed new algorithms which would be able to use these new forms of approximation in attacks, and devised the statistical frameworks allowing us to calculate the attack complexities, before designing the cost functions with these facts in mind.

We have also built on existing work in evolving nonlinear approximations not merely by incorporating a more detailed knowledge of the problem domain, but by studying the various possible move functions and by establishing the existence of a smooth search landscape for one type of move function when evolving nonlinear approximations.

We have also incorporated the newly evolved approximations into attacks on DES and Serpent, and although we have not improved on the performance of the best attack on DES, we have succeeded in devising an attack on 11-round Serpent with better data complexity than any other known-plaintext attack, and have also achieved the best time complexity of any attack so far on 11-round Serpent-256.

We now consider directions in which this research might proceed further.

Instead of trying to modify the approximations from existing linear attacks - with the resulting increase in the number of active plaintext/ciphertext bits - a promising research direction could be to develop new algorithms to search for approximations which achieve a better bias/active-outer-round-S-box tradeoff. Given the low data complexity of the differential-linear attacks on reduced-round Serpent, another promising research avenue might be to develop statistical frameworks for, and working prototypes of, differential-nonlinear attacks.

We mentioned earlier that we did not know how to combine multidimensional linear approximations and nonlinear approximations in the same attack. This may prove a promising research avenue, as might attempts to move from what is basically a multiple-

approximation attack using the multiple "related" nonlinear approximations to a multidimensional nonlinear attack utilising all linear combinations of the relateds, and indeed to explore further generalisations of multiple nonlinear cryptanalysis to forms of multidimensional cryptanalysis.

Although it is not in general possible to link together nonlinear approximations to the inner rounds of a cipher, for certain weak key classes it may be possible to do so in certain cases, leading to increasingly powerful attacks for such keys. Search algorithms to find such approximations may also prove useful in identifying weak key classes of this form.

We have already mentioned that use of the log-likelihood ratio statistic - or indeed approximations thereof - to achieve attacks with even lower data complexity is impaired for approximations more than a certain number of rounds in length by the linear hull effect. Collard et al.'s work [97] shows the true distribution for the bias of a single linear approximation becoming increasingly key-dependent, and diverging increasingly from the theoretical distribution calculated beforehand, as the number of approximated rounds increases. Research into means by which the scale of the effect might be estimated, and partial information about the theoretical distribution incorporated into a modified LLR-like statistic could prove fruitful. Possibly, for individual attacks, nonlinear approximations to inner rounds might be utilised to obtain information about the various key-dependent distributions.

There has also been research, as mentioned earlier [67, 201] into a variation of linear cryptanalysis *exploiting* the linear hull effect [145, 114, 200]. Although we have not yet found a way to exploit nonlinear approximations in the best-known use of this method, to whit the attack on PRESENT mentioned previously, research into combining nonlinear approximations with linear hull cryptanalysis could prove promising.

Chapter 7

Evaluation and conclusions

In this thesis, we set out to investigate the following hypothesis:

"That metaheuristic search techniques have a great deal of potential within cryptology; both to improve upon the design of modern-day ciphers and their components, and to contribute to their cryptanalysis."

In the conclusions to each of the individual technical sections, we have discussed the achievements of the corresponding research project in relation to this hypothesis, as well as identifying avenues for future research. We now consider these as a whole.

7.1 Single-output Boolean functions (Chapter 4).

In this section, we obtained Boolean functions suitable for use as the filter functions of LFSR stream ciphers, with optimal resistance to algebraic and fast algebraic attacks, and superior resistance to the other known types of attack compared to any existing construction for this type of function.

This work, however, possessed two principal limitations. First of all, the question of how to implement an evolved function, other than by storing its truth table in memory (possibly requiring more memory than would be available in a low-resource environment) is vital if these are to be utilised. The most promising research avenue in terms of addressing this would be for theoreticians to investigate the new functions to see if any can be transformed into functions of a (new or known) "type", such as the Carlet-Feng type [55], with equivalent properties and some known means of efficient implementation for any number n of input bits. If so, and if this were an existing type, these functions would have

superior cryptographic properties to any known functions of said type.

The second limitation was as follows: As the number of input bits increases, the time and memory complexity of the search algorithm increases exponentially, due to the properties of the known algorithms to calculate algebraic immunity and fast algebraic resistance. For more than 15 input bits, we were no longer achieving results superior to those of constructed functions, and a modest increase in the number of input bits would soon have required a very large number of gigabytes of memory indeed.

Again, the most promising research avenue in terms of addressing this would be to find some larger class of functions containing at least some of the newly evolved functions, such that some efficient means of constructing functions of this class for arbitrary values of n exists. For example, to construct a Carlet-Feng function, the cryptanalyst chooses some primitive element α of the field $GF(2^n)$, and, using finite field arithmetic, calculates the positions of the 1s in the truth table as follows: $\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^{n-1}-2}\}$. If any of the new functions, or a property-preserving transform thereof, were to be shown to be of the Carlet-Feng type, further research into the properties of these functions which lead to their possessing higher nonlinearity than others of this type, and the values of α corresponding to these, would be the next logical step.

7.2 Vectorial Boolean functions (Chapter 5).

Although metaheuristics have been applied to the evolution of S-boxes before, this was their first application to finding S-boxes with good differential uniformity. We also compared simulated annealing, which was applied to the evolution of S-boxes with different properties, with two metaheuristics which had not hitherto been applied to the evolution of vectorial Boolean functions, and carried out a comprehensive examination of various parameter choices for all three techniques.

Since metaheuristics have not previously been applied to evolving vectorial Boolean functions with good differential uniformity, we do not have any prior use of search techniques in this field to compare our work to. Comparing it with results constructed by theoreticians, for sizes greater than or equal to $n = 6$, we have not managed to match the best-obtained results.

This is in part due to the chief limitation we encountered - the "combinatorial explosion" in the size of the search landscape is extremely rapid, since this size is equal to $2^n!$ (the ! signifies factorial).

We have conducted, as may be seen in Appendix B, additional research into a potential means of mitigating this problem. This has not proven successful, but further research in this area should not be ruled out.

Another possibility for future research might be to look at ways to apply a "divide-and-conquer" methodology to the problem - for example, evolving two subfunctions with low differential uniformity that would then join up into the larger function. This would not guarantee that the main function had low differential uniformity, but since it would be necessary for, say, two 5×6 S-boxes to have differential uniformity 2 if their 6×6 concatenation was to, it might lead to a way of tackling a smaller search space to solve the problem, or produce S-boxes closer to the optimum.

In the meantime, however, it does not seem likely that the direct application of search to this problem will compete with the best results obtained through theoretical constructions in the short term.

7.3 Nonlinear approximations and nonlinear cryptanalysis (Chapter 6).

In this section, we have not simply evolved nonlinear S-box approximations. Through careful considerations of the differences between these and the linear approximations they would replace, we were able to define the properties they would need to have, the ways in which cryptanalytic algorithms utilising them would have to differ from conventional linear cryptanalysis, and the new statistical frameworks for these algorithms. All of this allowed us to evolve nonlinear approximations with superior properties to the best linear approximations for various cipher S-boxes.

We were also able to go further, joining our nonlinear approximations to linear approximations for the inner rounds of two ciphers, and evaluating the effect of this on the existing attacks on these ciphers. In the case of one of these ciphers, the new approximations improved in two different ways on the cryptanalytic state of the art for that cipher, demonstrating the power of metaheuristics in this form of cryptanalysis.

We have also addressed inadequacies with previous research using conventional linear cryptanalysis, and used more rigorous statistical reasoning both in assessing the performance of attacks with the evolved approximations and in comparing them to previous attacks.

The limitations we encountered fell into two main categories; those also encountered by conventional linear cryptanalysis without metaheuristics, and those unique to cryptanalysis with the newly-evolved nonlinear approximations.

The first category A limitation in the first category was the so-called "linear hull effect". This prevented us from employing the optimal statistic for distinguishing between two probability distributions. Future research into the linear hull effect, in particular on ways to estimate the scale of the effect for cipher approximations, would be key to making progress towards overcoming this.

The second category In some statistical models, nonlinear cryptanalysis has to deal with sets of variables that may not be independent - the so-called "related approximations" - and these statistical models are only valid when the variables are independent. Finding a way to quantify the effect of statistical dependence in this area would seem to be an important avenue for future research. This may well represent a significant unsolved problem within statistics in general, not merely in the context of nonlinear approximations, but it is certainly preventing the design of cost functions capable of addressing this issue.

Other areas for future research In linear cryptanalysis, several individual S-box approximations must be combined into an approximation of one cipher round, and several such approximations combined to approximate several of the inner rounds of a particular cipher. Searching for such multi-round approximations is a non-trivial task. Perhaps the most promising research avenue for nonlinear cryptanalysis would be the design of algorithms to automate the search for such approximations, albeit with nonlinear components in their outer rounds. Currently, we have to modify existing multi-round linear approximations, and the changes made to these to accommodate nonlinear approximations interfere with the fine-tuning done to keep the number of outer-round S-boxes down. Furthermore, it may well be that more high-bias nonlinear approximations could have been added to linear approximations specifically constructed with this in mind.

Some of the best cryptanalytic results against the Serpent cipher were obtained by using the "differential-linear cryptanalysis" technique. Using chosen-plaintext and chosen ciphertext structures such that N plaintext-ciphertext pairs could be used to provide in excess of $N/2$ samples for the attack, the researchers using this methodology were able to significantly decrease the data complexity - at the cost of making it more difficult to

obtain the required data.

It should be possible to use evolved approximations in a form of differential-*non*linear cryptanalysis. Since the linear approximations used in the existing differential-linear attacks could not be easily modified to accommodate nonlinear components, this might also require the researcher to devise new differential characteristics and linear approximations for the remaining rounds, which could then be joined to the nonlinear approximations. Nevertheless, this could lead to improved cryptanalysis of eleven or more rounds of Serpent with lower CC/CP data complexity than hitherto.

Multidimensional linear cryptanalysis has been seen to be a powerful variant of the basic linear cryptanalysis technique. One possible research direction would be a search for ways to incorporate nonlinear approximations into this. Moreover, at the end of Chapter 6, we briefly described various possible "multidimensional nonlinear" cryptanalysis techniques. It may well be worth investigating these to see if they can lead to improvements in the attack's data complexity that would outweigh their increased time complexity.

Combining nonlinear cryptanalysis with the so-called "linear hull cryptanalysis" has not proven possible in the case of the linear hull attack against the cipher PRESENT, but may be possible in the case of other such attacks.

In the multi-round approximations used in linear and nonlinear cryptanalysis, we have so far only been able to evolve nonlinear S-box approximations for the first and last rounds of these. Using nonlinear approximations in other rounds is not so straightforward; it is known that improvements resulting from these would probably apply only to some of the possible keys attacked [130], but research into weak keys of this nature, and perhaps whether all keys might be weak depending on which of a choice of various nonlinear approximations is used, could lead to improved performance when inner-round nonlinear approximations (which could be nonlinear at both the input and output ends, not simply linear at one end and nonlinear at the other) are incorporated into an attack. We believe that the use of inner-round nonlinear approximations may also be of use in researching the extent of the linear hull effect and how it varies depending on the attacked key.

7.4 A unified view...

We have attempted not only to argue that metaheuristic search has a role in modern cryptology, but to actively place it within the cryptanalyst's (figurative) toolbox. In some areas of cryptology we have clearly succeeded in demonstrating the potential of

metaheuristics, in some areas we have had a more modest success of this form, and in some areas we have had to accept that metaheuristic search may not be the most effective technique in the short term.

We have attacked larger-scale, more sophisticated problems in cryptology than have previously had metaheuristics applied to them, and have managed to advance the limits of cryptologic knowledge and the state of the art in various areas. We believe that we have provided strong evidence in favour of our hypothesis, and hope that this will inspire further research into the use of metaheuristics in cryptology.

In considering areas for future research, a common theme has been the fact that metaheuristics should not replace previous techniques completely, but rather combine with them. For example, nonlinear cryptanalysis has combined metaheuristically-evolved "nonlinear approximations" with the existing technique of linear cryptanalysis and the constructions designed by other means for use in that area. Similarly, evolving filter functions can only go so far in terms of the number of input bits it can handle, but it has shown that superior properties can be achieved to those of the existing functions, and has made it possible to look for classes of Boolean functions which have such properties and which exist regardless of the value of n. We think that cryptanalysts should look for areas in their work in which they can use metaheuristics, and hope that in the next few years the use of metaheuristics as part of the cryptanalyst's toolbox will become a matter of routine.

Appendix A

Truth tables of evolved filter functions

We present, in hexadecimal format, some of the truth tables of the evolved functions. Any researchers who would like the full set of evolved functions with nonlinearities as shown in Tables 4.1 and 4.2 are welcome to contact the authors directly (`jmclaugh@cs.york.ac.uk`).

$n = 6$**:** The following two truth tables are representatives of the discovered equivalence classes:

`3502 8c3e f607 f571`

and

`385d b3b3 6f90 58a1`

$n = 7$: The representative truth tables are:

`094f ddf3 299f 8b6c 15a4 42c7 5185 edc8`

and

`58ff 2d3a d029 4127 1958 f4d9 d436 3b53`

$n = 8$:

`fbf2 6023 2e62 c9c7 aec4 d8b6 e4b2 ade5`
`616e 3c45 03f3 08d5 5baf e9aa 9609 6031`

possesses fewer 24s (the maximal absolute value) in its Walsh spectrum than any

other annealed function of this size, and

```
cccf 5ed2 8ac9 2b6a 7470 ceea fea5 3b90
4f42 50bd c031 6091 bee7 a079 e2c7 4368
```

has the most zeroes in said spectrum.

$n = 9$:

```
94f2 066e 9763 7bf4 5c73 be75 c0ff 98fa
828f ba5b 4411 23b8 f288 b8e7 958c bda6
d92d d595 123d 343c ed40 3832 2233 e129
81b5 e652 349f 8140 d005 bf79 ad1e ae17
```

has the fewest 36s in its spectrum, and

```
d39e f4ef 1781 c8ba 2cdb 34d0 f6e7 83cf
8a17 d712 bb5a 51d0 d762 6825 ab33 a1dd
b4e3 a66a e2c0 a722 0ecf 154d 2181 a63e
bc0f 09f0 14d8 88e4 bba5 3679 77b0 46d0
```

has the most zeroes.

$n = 10$:

```
67d7 20ef af0f 51a5 bdfd 440b 4080 4bf4
a543 b21f 7796 de5f 31fc 998f 553b bbaa
c7c0 c16c 3509 a835 894f 1589 499f 10d6
219f dfbc b50a bf3a cd89 24d6 69d5 461e
ebcb cba6 03b4 0313 d37c ea74 4d88 97aa
176c 7176 020b 8d98 315b 69e1 2a8b 4a9a
87b6 66d8 1c09 0347 5137 e400 7f2a 34d9
f34d d885 f7b3 d654 f213 ac31 a4f1 82b1
```

has fewer 52s in its spectrum than any other annealed function of this size and nonlinearity.

Appendix B

The proof of Theorem 5.1.18

We show that given any bijection S over $GF(2^n)$, an affine-equivalent bijection S_2 exists such that S_2 maps 0 and all 2^i to themselves (i.e. all $(n+1)$ values with Hamming weight < 2 are fixed points), and all $2^i + 1$ to values within a certain restricted range. We then show that for S with differential uniformity 2 or 4, the range of output values for certain inputs can be made even narrower. We do not succeed, however, in obtaining a unique representation - or "normal form" for the affine equivalence class of S in this way; and the ramifications of this when trying to evolve bijections over $GF(2^n)$ (such as cryptographic S-boxes) using memetic or ant-based metaheuristic algorithms are considered.

B.0.1 Preliminaries.

Lemma B.0.1. *Let S denote a bijective S-box. It is trivial to construct another $n \times n$ S-box, S_2, which is EA-equivalent to S and which maps all inputs with Hamming weight ≤ 1 to themselves. However, S_2 is not necessarily bijective.*

Proof. Let $S_2 = S \oplus Cx \oplus d$. Let d be the bitstring representation of $S(0)$. By choosing the matrix C so that it will map every input x with Hamming weight 1 to $S(x) \oplus S(0) \oplus x$, we obtain an S_2 with the properties described.

Such a C can be constructed by letting the ith column be the bitstring representation of $2^{n-i} \oplus S(2^{n-i}) \oplus S(0)$.

We now present an example where S_2 is constructed as described above from some bijective S, but is not itself bijective. Let S be the S-box from the Courtois Toy Cipher [106]:

0	1	2	3	4	5	6	7
7	6	0	4	2	5	1	3

The matrix C must then be defined as follows:

$$C = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Note that $S(0) \oplus C \cdot 0 = 7 \oplus 0 = 7$. However, $S(7) \oplus C \cdot 7 = 3 \oplus 4 = 7$.
It follows that $S_2(0) = S_2(7) = 7 \oplus d$. (Since d is defined as $S(0)$, $d = 7$.) □

In the following section, we shall prove that at least one S-box affine-equivalent to S, and mapping all inputs with Hamming weight ≤ 1 to themselves as described, but which is also bijective, must always exist. We shall describe a procedure to construct this S-box from S; firstly by constructing an S-box mapping 0, 1 and 2 to themselves, and then applying a more complicated procedure to construct the final S-box from this. Although we do not manage to prove that any other values can be mapped to themselves without jeopardising affine equivalence, we do limit the range of values that can be mapped to by some of the S-box inputs with Hamming weight 2.

B.0.2 Constructing the equivalent bijection.

We shall construct a sequence of S-boxes, the first of which we shall construct from the original S-box, and will construct each successive S-box from its predecessor until we obtain one with the desired properties.

Lemma B.0.2. *Let S_1 be a bijective S-box.*

Then there exists at least one bijective S-box S_2 with $S_2(0) = 0$, and which is also affine-equivalent to S_1.

Proof. To construct S_2, we can either

- Xor every output of S_1 with $S_1(0)$, or
- Let S_2 be defined as $S_2(x) = S_1(x \oplus S_1^{-1}(0))$.

□

Lemma B.0.3. *If S_1 has differential uniformity $< 2^n$, there exist at least two S_2 equivalent to S_1 with the properties described above.*

Proof. If $S_1(0) = 0$, let S_a be some affine-equivalent S-box such that this is not the case. Perhaps we could xor all of S_1's outputs with 1 to achieve this. If $S_1(0) \neq 0$, let $S_a = S_1$.

Let S_{2a} be the box we obtain by xoring every output of S_a with $S_a(0)$.

Let S_{2b} be the box defined by $S_{2b}(x) = S_a(x \oplus S_a^{-1}(0))$.

(Note that one of these may be the same S-box as the original S_1.)

If S_{2a} and S_{2b} were the same S-box, it would follow that $S_a(x \oplus S_a^{-1}(0))$ was equal to $S_a(x) \oplus S_a(0)$ in all cases. The entry in $S_a's$ difference distribution table for row $S_a^{-1}(0)$ and column $S_a(0)$ would then be equal to 2^n. However, since differential uniformity is affine-invariant, this would lead to a contradiction, as S_1 does not have differential uniformity 2^n. □

If n is equal to 1, then this is enough to prove the main result, since the bijectivity of the constructed S-box would mean that it also had to map 1 to itself. We shall therefore assume from here on that $n \geq 2$.

Lemma B.0.4. *For any nonzero n-vector, there exists at least one linear bijection expressed as an $n \times n$ matrix over $GF(2)$ such that said vector is a column thereof.*

Furthermore, for any distinct pair of nonzero n-vectors, there exists at least one linear bijection expressed as an $n \times n$ matrix over $GF(2)$ such that each of these vectors is a column thereof.

Proof. The result is trivial. The only condition imposed on the matrix by its bijectivity is that it must be invertible, which is the case if and only if all its columns are linearly independent. This does not prevent us from choosing the first column arbitrarily (as long as it is nonzero), and the only restriction imposed on the second column is that it should not be equal to zero or to the first column. We choose the rest of the columns accordingly, and then reorder them if we do not wish the vector (or vectors) we started with to occupy the first column (or first two columns.) □

Corollary B.0.5. *For all nonzero $x \in GF(2^n)$, and for any $0 \leq k < n$, there exists at least one bijective matrix M such that $M(2^k) = x$.*

Proof. Simply choose M as described above so that its $(n-k)$th column is the vector x. □

Corollary B.0.6. *For all distinct nonzero $(x, y) \in GF(2^n)$, and for any $0 \leq k < l < n$, there exists at least one bijective matrix M such that $M(2^k) = x$ and $M(2^l) = y$.*

Proof. Choose M so that its $(n-k)$th column is x and its $(n-l)$th column is y.

Note that $M(2^k \oplus 2^l)$ will be equal to $(x \oplus y)$. □

Corollary B.0.7. *For all nonzero $x \in GF(2^n)$, and for any $0 \leq k < n$, there exists at least one bijective matrix M such that $M(x) = 2^k$.*

Proof. Choose some N as described in Corollary B.0.5 such that $N(2^k) = x$. Then N^{-1} is M as desired. □

Corollary B.0.8. *For all distinct nonzero $(x, y) \in GF(2^n)$, and for any $0 \leq k < l < n$, there exists at least one bijective matrix M such that $M(x) = 2^k$ and $M(y) = 2^l$.*

Proof. Choose some N as described in Corollary B.0.6 such that $N(2^k) = x$ and $N(2^l) = y$. Then N^{-1} is M as desired.

Note that since $N(2^k \oplus 2^l)$ will equal $x \oplus y$, $M(x \oplus y) = (2^k \oplus 2^l)$. □

Theorem B.0.9. *Let S be a bijective S-box such that $S(0) = 0$. For any $0 \leq k < l < n$, there exists at least one affine-equivalent bijective S-box S_2 such that:*

- $S_2(0) = 0$,
- $S_2(2^k) = 2^k$,
- $S_2(2^l) = 2^l$, *and*
- $S_2(x) = M \cdot S(x)$ *for some linear bijective matrix M.*

Proof. Any bijective linear transformation of 0 is 0, so $S_2(0) = 0$.

From Corollary B.0.8, we can choose M to be a transformation mapping $S(2^k)$ to 2^k and $S(2^l)$ to 2^l. (Again, we note that M will also map $(S(2^k) \oplus S(2^l))$ to $(2^k \oplus 2^l)$.) □

Theorem B.0.10. *Let S be a bijective S-box such that $S(0) = 0$. For any $0 \leq k < l < n$, there exists at least one affine-equivalent bijective S-box S_2 such that:*

- $S_2(0) = 0$,
- $S_2(2^k) = 2^k$,

- $S_2(2^l) = 2^l$, *and*
- $S_2(x) = S{\cdot}M(x)$ *for some linear bijective matrix* M.

Proof. Any bijective linear transformation of 0 is 0, so $S_2(0) = 0$.

From Corollary B.0.6, we can choose M to be a transformation mapping 2^k to $S^{-1}(2^k)$ and 2^l to $S^{-1}(2^l)$. (Again, we note that M will also map $(2^k \oplus 2^l)$ to $(S^{-1}(2^k) \oplus S^{-1}(2^l))$.) □

We see that it is fairly straightforward to construct an affine-equivalent bijective S-box mapping 0, 1, and 2 to themselves. If $n = 2$, we have now achieved the desired result, so we shall now assume $n \geq 3$.

Lemma B.0.11. *Let* S *be an APN S-box mapping 0, 1 and 2 to themselves.* $S(3)$ *cannot be equal to 3.*

Proof. If $S(3)$ were equal to 3, (input difference 3, output difference 3) would occur for the input pairs (0, 3), (3, 0), (1, 2), and (2, 1), meaning that the differential uniformity of S would be at least 4 and contradicting the statement that it is APN. □

We note that for values of n such as 4, for which APN bijections do not exist [173], we cannot guarantee that a given S-box mapping 0, 1 and 2 to themselves will not also map 3 to itself, but it seems highly unlikely that boxes mapping 0, 1, 2 and 3 to themselves will not be in the minority. Certainly APN bijective S-boxes are known to exist for $n = 6$ [45] and for all odd n [197]; furthermore differentially-4-uniform S-boxes are known to exist for all n [197] and a later result in the appendices to this document will show that, given some bijective differentially-4-uniform S-box mapping 0, 1, 2 and 3 to themselves, we can construct from it an affine-equivalent differentially-4-uniform bijective S-box mapping 0, 1, 2 and 3 to 0, 1, 2 and 5 respectively.

Theorem B.0.12. *Let* S *be a bijective S-box mapping 0, 1, 2 to themselves, but not mapping 3 to itself. There exist bijective S-boxes* S_1, S_2, S_3 *affine-equivalent to* S *which also map 0, 1, and 2 to themselves, such that:*

- $S_1(3) = 5$
- $S_2(3) = 6$
- $S_3(3) = 7$

Proof. If $S(3) \leq 7$, we can construct the desired S_i immediately, by applying a process which we will refer to as "controlled-XOR":

Definition B.0.13. Let $CXOR(i, a_1 a_2 \ldots a_{i-1})$ be a matrix identical to the identity matrix except in its ith-last column

Instead of

$$\begin{vmatrix} 0 \\ \ldots \\ 0 \\ 1 \\ 0 \\ 0 \\ \ldots \\ 0 \end{vmatrix}$$

let it be

$$\begin{vmatrix} 0 \\ \ldots \\ 0 \\ 1 \\ a_1 \\ a_2 \\ \ldots \\ a_{i-1} \end{vmatrix}$$

Applying $CXOR$ to the outputs of the S-box will map all values $< 2^{i-1}$ to themselves, as also any values $> 2^{i-1}$ whose ith-last bit is zero. $CXOR$ will map any output values with nonzero ith-last bit from

$$\begin{vmatrix} c_1 \\ \dots \\ c_p \\ 1 \\ b_1 \\ b_2 \\ \dots \\ b_{i-1} \end{vmatrix}$$

to

$$\begin{vmatrix} c_1 \\ \dots \\ c_p \\ 1 \\ b_1 \oplus a_1 \\ b_2 \oplus a_2 \\ \dots \\ b_{i-1} \oplus a_{i-1} \end{vmatrix}$$

and we simply choose $(a_1, \dots, a_{i-1})$ to give the desired result.

In this case, we simply apply $CXOR$ with i=3 and a_1, a_2 chosen to give whichever of 5, 6 and 7 is desired as the value for $S(3)$.

If $S(3) > 7$, we first create an S-box S_a mapping 0, 1 and 2 to themselves and such that $S_a(3) \leq 7$.

This will involve applying a matrix which we will refer to as MSB_SHIFT; which is similar to the matrices which comprise the $SWAP$ and $CNOT$ quantum gates.

Definition B.0.14. Let $MSB\_SHIFT(S(i))$ be a matrix identical in all but two rows to the identity matrix. These rows must be: the row in which the first (i.e. the MSB - the topmost when it is expressed as a column vector) nonzero bit of $S(i)$ occurs, and the row immediately below it. Let them be denoted $R1$ and $R2$ respectively.

Whereas the identity matrix includes:

```
ROW ABOVE R1:  0...1000...0
          R1:  0...0100...0
          R2:  0...0010...0
ROW BELOW R2:  0...0001...0
```

the corresponding rows of $MSB\_SHIFT(S(i))$ are:

```
ROW ABOVE R1:  0...1000...0
          R1:  0...0A10...0
          R2:  0...01X0...0
ROW BELOW R2:  0...0001...0
```

where:

- A is the value of the bit immediately following the first nonzero bit of $S(i)$ (so $S(i)$ is of the form $1Abc...\omega$)
- X is either 0 or $(1 \oplus A)$.

It is easy to confirm that the columns of $MSB\_SHIFT(S(i))$ are all linearly independent.

We transform S by applying $MSB\_SHIFT(S(i))$ to the outputs of S; i.e. by calculating
$MSB\_SHIFT(S(i))(S)$. The effect of $MSB\_SHIFT(S(i))$ on $S(i)$ (indeed, on all outputs of S) is to map

$$\left|\begin{array}{c} 0 \\ \dots \\ 0 \\ 1 \\ A \\ b \\ \dots \\ \omega \end{array}\right|$$

to

$$\left|\begin{array}{c} 0 \\ \dots \\ 0 \\ (A{\cdot}1) \oplus (1{\cdot}A) = 0 \\ (1{\cdot}1) \oplus (X{\cdot}A) = 1 \\ b \\ \dots \\ \omega \end{array}\right|$$

and

$$\left|\begin{array}{c} 0 \\ \dots \\ 0 \\ 0 \\ 1 \\ b \\ \dots \\ \omega \end{array}\right|$$

to

$$\begin{vmatrix} 0 \\ \cdots \\ 0 \\ (A{\cdot}0) \oplus (1{\cdot}1) = 1 \\ (1{\cdot}0) \oplus (X{\cdot}1) = X \\ b \\ \cdots \\ \omega \end{vmatrix}$$

To obtain an S-box S_η where $S_\eta(3) \leq 7$ and which maps 0, 1 and 2 to themselves, we apply $MSB\_SHIFT(S(3))$ to S to create a new S-box, S_a. If $S_a(3) > 7$, we create and apply $MSB\_SHIFT(S_a(3))$ to S_a to obtain S_b... Eventually, this procedure will yield S_η as desired.

We can then use on S_η the procedure we would have used on S had $S(3)$ been less than or equal to 7. □

Theorem B.0.15. *Let S be a bijective S-box mapping 0, 1, 2 to themselves, with differential uniformity ≤ 4. There exist bijective S-boxes S_1, S_2, S_3 affine-equivalent to S which also map 0, 1, and 2 to themselves, such that:*

- $S_1(3) = 5$
- $S_2(3) = 6$
- $S_3(3) = 7$

Proof. If S is APN, or if $S(3) \neq 3$, we have already obtained the desired result. Let us therefore focus on the case where S is differentially-4-uniform and $S(3) = 3$.

Apply a matrix M_a to the S-box outputs mapping 2 to some value $x \geq 4$, but preserving the property that 0 and 1 are mapped to themselves. This will map 3 to $(x \oplus 1)$.

Let the S-box resulting from this be denoted S_a. We have $S_a(2) = x$, $S_a(3) = (x \oplus 1)$.

The input difference between $S_a^{-1}(2)$ and $S_a^{-1}(3)$ cannot be equal to 1. For if it could, we would have (input difference 1, output difference 1) for $((x_1, x_2), (y_1, y_2)) =$

- $((0, 1), (0, 1))$
- $((2, 3), (x, x \oplus 1))$

- $((S_a^{-1}(2), S_a^{-1}(3)), (2, 3))$

contradicting our assertion that S has differential uniformity ≤ 4.

Apply some matrix M_b to the S-box inputs, mapping 2 to $S_a^{-1}(2)$ and 1 to itself. This will map 3 to $(S_a^{-1}(2) \oplus 1)$, which we have already shown cannot be $S_a^{-1}(3)$.

We can now continue with the procedure described in Theorem B.0.12 to achieve our desired result. □

Without loss of generality, we shall henceforth assume that the preferred value for $S(3)$ was 5.

The procedure now starts to become more complicated. The purpose served by Theorem B.0.16 will not be obvious until we reach Theorem B.0.17. These two theorems will allow us to construct an equivalent S-box mapping all inputs with Hamming weight 0 or 1 to themselves, and 3 to 3 or 5 (we will assume that 3 is mapped to 5 for an S-box with differential uniformity 2 or 4). We will then demonstrate how, after an S-box mapping 0, 1, 2, and 4 to themselves, and 3 to 3 or 5, has been constructed, assuming $n \geq 4$, we can construct another equivalent S-box as described but with additional restrictions on the values of $S(x)$ for x of the form $2^i + 1$, using a procedure which will reduce the number of times we need to apply that of the below theorem:

Theorem B.0.16. *Let S be a bijective S-box such that, for some $h \geq 1$,*

- $S(x) = x \ \forall \ x \in \{0, 1, 2, \ldots, 2^h\}$,
- $S(3) = 3$ *or* 5,
- $S^{-1}(2^{h+1}) < 2^{h+1}$, *and*
- $n \geq (h + 2)$.

There exist at least $(h + 3)$ bijective S-boxes S_2 affine-equivalent to S such that

- $S_2(x) = x \ \forall \ x \in \{0, 1, 2, \ldots, 2^h\}$,
- $S_2(3) = S(3) = 3$ *or* 5,
- $S_2^{-1}(2^{h+1}) \geq 2^{h+1}$, *and*
- $S_2(x) = M(S(x))$ *for some linear bijection M.*

Proof. We note first of all that the procedure described in this proof will not be necessary for $h = 1$, since under the circumstances described $S^{-1}(4)$ must be greater than or equal to 4. We can therefore replace the assumption that $h \geq 1$ with the assumption that $h > 1$.

The matrix M is in fact a $CXOR$ matrix as described in Definition B.0.13. Let all of M's columns except the $(n - (h + 1))$st be identical to the corresponding column of the identity matrix. Since they are linearly independent, this is permissible.

This column must be of the form

$$\begin{vmatrix} 0 \\ \dots \\ 0 \\ 1 \\ x_1 \\ x_2 \\ \\ \dots \\ \\ x_{h+1} \end{vmatrix}$$

where at least one x_i should be nonzero, and such that the column should not be the same as any S-box outputs x with the property that $S^{-1}(x) < 2^{h+1}$. From the condition that at least one x_i should be nonzero, we see that there are at most $2^{h+1} - 1$ possible columns. It should be clear that this column is also linearly independent of the other columns in the matrix.

We look at the question of how many x such that $S^{-1}(x) < 2^{h+1}$ could be equal to such a column. There are 2^{h+1} values of x such that $S^{-1}(x) < 2^{h+1}$.

$(h+1)$ of these are the values $\{2^0, 2^1, \dots, 2^h\}$. None of these could equal such a column.

An $(h + 2)$nd such x which cannot correspond to the column described is 0.

An $(h + 3)$rd such value of x is $S(3)$. (This is easily shown to follow from the fact that $h > 1$.)

We obtain an $(h + 4)$th such value by noting that $S^{-1}(2^{h+1}) < 2^{h+1}$, and that this cannot correspond to the column due to the condition that at least one x_i be nonzero.

We see that at most $(2^{h+1} - (h+4))$ of the first 2^{h+1} truth table entries can correspond to such columns, leaving at least $(2^{h+1} - 1) - (2^{h+1} - (h + 4)) = (h + 3)$ valid choices for

the column of M.

How does this work? The transformation $S_2 = M(S(x))$ will not affect any outputs of S which are $< 2^{h+1}$. It will, however, map the S-box output equal to the column to 2^{h+1}. The fact that this output corresponds to an input greater than or equal to 2^{h+1} is what causes the transformation to obtain the desired result.

Each different choice for this column will result in a different value for $S_2(S^{-1}(2^{h+1}))$ (and a different value for $S_2^{-1}(2^{h+1})$), hence our statement that at least $(h+3)$ such S-boxes exist. □

Theorem B.0.17. *Let S be a bijective S-box such that*

- $S(x) = x$ *for* $x \in \{0, 1, 2, \ldots, 2^h\}$ *for some* h,
- $S(3) = 3$ *or* 5,
- $S^{-1}(2^{h+1}) \geq 2^{h+1}$, *and*
- $n \geq (h + 2)$.

There exists at least one bijective S-box S_2 affine-equivalent to S such that $S_2(x) = x$ for all $x \in \{0, 1, 2, 2^2, \ldots, 2^h, 2^{h+1}\}$, $S_2(3) = S(3)$ and $S_2(x) = S(M(x))$ for some linear bijection M.

Proof. If $S^{-1}(2^{h+1}) = 2^{h+1}$, we do not need to do anything. Otherwise, let M be such that its $(n-(h+1))$st column is equal to $S^{-1}(2^{h+1})$, and such that its $(n-i)$th column is equal to 2^i (i.e. equal to the corresponding column of the identity matrix) for all $0 \leq i \leq h$. All other columns can be chosen arbitrarily, as long as they are linearly independent of these and of each other.

M will map 2^{h+1} to $S^{-1}(2^{h+1})$, and will leave invariant all values $< 2^{h+1}$. Hence S_2 will map such values to the same outputs as S.

Since $S^{-1}(2^{h+1}) \geq 2^{h+1}$, it cannot be in the spanning set of the columns to the right of it, and hence M is a valid linear bijection. □

It follows from the preceding results that:

Corollary B.0.18. *Every bijective S-box S is affine-equivalent to at least one bijective S-box S_2 such that S_2 maps all inputs with Hamming weight < 2 to themselves, and 3 to either 3 or 5. Furthermore, if S has differential uniformity of 4 or less, we may assume that $S_2(3) = 5$.*

We will now show how a variation on this procedure may be obtained to construct an S-box with the properties described and with further restrictions on the values mapped to by some of the x with Hamming weight 2.

Theorem B.0.19. *Let S be a bijective S-box such that, for some $h \leq (n-2)$, $S(x) = x$ for all $x \in \{0, 2^0, 2^1, \ldots, 2^h\}$, $S(3) \leq 5$, $S(2^i + 1) \leq (2^{i+2} - 1)$ for all $1 < i < h$, and such that $n \geq 3$.*

(The result does also trivially hold for $h = (n-1)$, but for this value of h the below procedure does not need to be applied.)

There exists at least one affine equivalent bijective S-box S_2 such that $S_2(x) = x$ for all $x \in \{0, 1, 2, 2^2, \ldots, 2^h\}$, $S_2(3) = S(3)$, and $S_2(2^i + 1) \leq (2^{i+2} - 1)$ for all $1 < i \leq h$.

Proof. If $S(2^h + 1) \geq 2^{h+2}$, we keep constructing and applying $MSB\_SHIFT(S(2^h + 1))$ until this has ceased to be the case.

MSB_SHIFT is only used if $S(2^h + 1) \geq 2^{h+2}$. It has no effect on any S-box output in which the first nonzero bit occurs two places or more later than the first nonzero bit of $S(2^h + 1)$. Hence, any S-box output $< 2^{h+1}$ is unaffected. This means that, for all $0 < i \leq h$, all $S(2^i) = 2^i$ are unaffected, $S(0)$ is unaffected, and for all $(0 < i < h)$, all $S(2^i + 1) \leq (2^{i+2} - 1)$, and so $\leq (2^{h+1} - 1)$, are unaffected.

We note that the use of MSB_SHIFT in this fashion will eventually result in a situation where $2^{h+1} \leq S(2^h + 1) \leq (2^{h+2} - 1)$. (Since no S-box output less than 2^{h+1} is affected, we know that 2^{h+1} is a lower bound for the value of $S(2^h + 1)$ at the end of this procedure.) □

Through the use of $CXOR$ on the MSB_SHIFTed value, we can reduce the upper bound for the $S(2^i + 1)$ even further. Let us begin with the specific case of $S(5)$:

Lemma B.0.20. *Let S be a bijective S-box such that $S(x) = x$ for $x \in \{0, 1, 2, \ldots, 4\}$, such that $S(3) \leq 5$ and such that $n \geq 3$.*

There exists at least one affine equivalent bijective S-box S_2 such that $S_2(x) = x$ for $x \in \{0, 1, 2, \ldots, 4\}$, $S_2(3) = S(3)$, $S_2(5) \leq 11$, and (if $S_2(5) > 8$), $S^{-1}(8) \geq 8$.

Proof. If $S(5) \leq 11$ and $S^{-1}(8) \geq 8$ already, we do not need to proceed any further. If $S(5) \leq 7$, again, we do not need to proceed further. We assume for the rest of the proof that $n \geq 4$, since the result is trivially true for $n = 3$.

If $S_2(5) > 11$, we begin by applying the procedure described in the proof of Theorem B.0.19, so that $8 \leq S_2(5) \leq 15$.

$S_2(5)$ will now be of the form

$$\left|\begin{array}{c} 0 \\ \dots \\ 0 \\ 1 \\ y_1 \\ y_2 \\ y_3 \end{array}\right|$$

We wish to use $CXOR(4, a_1a_2a_3)$ as described above to

- replace $S_2(5)$ with a vector of this form such that $y_1 = 0$, and
- to ensure that the procedure of Theorem B.0.16 will not need to be applied (as otherwise the application of said procedure might undo what we had achieved here.) (That is, we need to ensure $S_2^{-1}(8) \geq 8$).

We know that all of $S(0, 1, 2, 3, 4)$ are less than or equal to 7, but do not know if this is the case for $S(6)$ or $S(7)$. This gives us up to two values that may be of the form $1y_1ij$ and be such that using the corresponding y_1ij as $a_1a_2a_3$ for the $CXOR$ would result in $S^{-1}(8)$ being less than 8. A third such value arises from the need not to xor with $S_2(5)$ itself. As there are four y_1ij to choose from $(y_100, y_101, y_110, y_111)$, of which only three are potentially problematic, at least one suitable y_1ij for the $CXOR$ will always exist. □

Let us now generalise to the remaining $S(2^i + 1)$

Theorem B.0.21. *Let S be a bijective S-box such that, for some $h \leq (n-2)$ (The result does trivially hold for $h = (n-1)$, but the below procedure is neither necessary nor applicable in said case):*

- $S(x) = x \ \forall \ x \in \{0, 1, 2, 2^2, \dots, 2^h\}$
- $S(3) \leq 5$
- $S(2^i + 1) \leq (2^{i+2} - 2i - 1)$ *for all* $0 < i < h$,
- $n \geq 3$.

There exists at least one bijective S-box S_2 affine equivalent to S such that:

- $S_2(x) = x \ \forall \ x \in \{0, 1, 2, 2^2, \ldots, 2^h\}$,
- $S_2(3) \leq 5$,
- $S_2(2^i + 1) = S(2^i + 1)$, *and hence* $S_2(2^i + 1) \leq (2^{i+2} - 2i - 1)$, *for all* $0 < i < h$, *and*
- $S_2(2^h + 1) \leq (2^{h+2} - 2h - 1)$.

Proof. Consider first of all the special case $2^0 = 1$. $S(2^0+1) = S(2) = 2$. $(2^{0+2}-2\cdot 0-1) = (4 - 0 - 1) = 3$.

Consider also the particular cases $h = 1$ and $h = 2$. $S(2^1 + 1) = S(3)$, which we have already stated is less than or equal to 5. $(2^{1+2} - 2\cdot 1 - 1) = (8 - 2 - 1) = 5$.

$S(2^2 + 1) = S(5)$. $2^{2+2} - 2\cdot 2 - 1 = 11$, and we have already shown that we can always achieve $S(5) \leq 11$. (in fact, we will later show that $S(5) \leq 10$ can always be achieved for an APN.)

We see therefore that there is no contradiction inherent in the properties of S as described above. Let us therefore assume from here on that $h \geq 3$.

If $S_2(2^h + 1) \leq (2^{h+2} - 2h - 1)$ already, we do not need to proceed any further.

Otherwise, we begin by applying the procedure described in the proof of Theorem B.0.19, so that $2^{h+1} \leq S_2(2^h + 1) \leq (2^{h+2} - 1)$.

$S_2(2^h + 1)$ will now be of the form

$$\left|\begin{array}{c} 0 \\ \ldots \\ 0 \\ 1 \\ a_1 \\ a_2 \\ \ldots \\ a_{h+1} \end{array}\right|$$

We wish to use $CXOR(h + 2, b_1 b_2 \ldots b_{h+1})$ to

- replace this with another vector of this form such that $a_1 a_2 \ldots a_{h+1} \leq (2^{h+1} - 2h - 1)$, and

- to ensure that the procedure of Theorem B.0.16 will not need to be applied (as otherwise said procedure might undo all that we had achieved here.)

How many other S-box outputs $S(k)$ ($k < 2^{h+1}$) are there such that $2^{h+1} \leq S_2(k) \leq (2^{h+2} - 1)$? For $k < 8$, there are at most two (only $S(6)$ and $S(7)$ can, at this point, take values in such a range.) For larger k, there are at most $\sum_{j=3}^{h} 2^j - 2$ such outputs (since $S(2^j)$ and $S(2^j + 1)$, for each j, either do not at this stage take values in this range or, for $j = h$, are the value we wish to $CXOR$.)

This gives us a total of $(2 + \sum_{j=3}^{h} 2^j - 2)$ outputs which may restrict the range of values we can $CXOR$ with (lest they be mapped to 2^{h+1}). $2^2 - 2 = 2$, so in fact we have $\leq \sum_{j=2}^{h}(2^j - 2)$ such outputs. In fact, as $2^1 - 2 = 0$, we have less than or equal to

$$\sum_{j=1}^{h}(2^j - 2) = \sum_{j=1}^{h} 2^j - 2h = (2^{h+1} - 2) - 2h = 2^{h+1} - 2h - 2$$

such outputs.

In the worst-case scenario, these will prevent the $CXOR$s that would have resulted in the $(2^{h+1}-2h-2)$ smallest values $> 2^{h+1}$. Hence $S_2(2^h+1) \leq [(2^{h+1}+1)+(2^{h+1}-2h-2)] = 2^{h+2} - 2h - 1$. □

We can now obtain the following result:

Theorem B.0.22. *Every bijective S-box S is affine-equivalent to at least one bijective S-box S_2 such that S_2 maps all inputs with Hamming weight less than 2 to themselves, 3 to 5, 5 to some value ≤ 11, and all $2^i + 1$ $(3 \leq i \leq (n-1))$ to some value $\leq 2^{i+2} - 2i - 1$.*

Proof. We shall begin by addressing the part of the result that states that $S(5) \leq 11$.

- Consider applications of Theorem B.0.17's procedure carried out after the point at which the procedure described in the proof of Lemma B.0.20 has either been carried out or deemed unnecessary. These will leave invariant the effect of S-box inputs less than 2^{h+1}, which by that point will always be ≥ 8. Hence, these will not affect the value of $S(5)$.

- If the procedure described in the proof of Lemma B.0.20 was carried out, the procedure described in the proof of Theorem B.0.16 will not be carried out until $2^{h+1} \geq 16$ (in fact, due to the procedures in the proofs of Lemma B.0.20 and Theorem B.0.21,

it might not be carried out until much later or indeed at all). As $11 < 16$, and as it was stated in the proof of Theorem B.0.16 that no S-box outputs $< 2^{h+1}$ would be affected by the procedure, it follows that the value of $S(5)$ remains unaffected.

- If this procedure was not carried out, and if the reason for this was that $S(5) \leq 7$, then Theorem B.0.16's procedure may be carried out when $2^{h+1} = 8$. However, since this is greater than $S(5)$ under these circumstances, the value of $S(5)$ remains unaffected.

We thus prove the part of the result pertaining to $S(5)$.

We now need to address the question of any other values of the form $(2^i + 1)$ that we have caused to be mapped to values $\leq (2^{i+2} - 2i - 1)$. For $i = (n - 1)$, this is always the case, and we need not consider the corresponding $(2^i + 1)$. For the remaining values of $i \geq 3$, it is possible that Theorem B.0.16's procedure may have to be applied, if the value of $S(2^i + 1)$ was too small to require the procedures of Theorem B.0.19 and Theorem B.0.21. If so, the value of $S(2^i + 1)$ will be too small to be affected by application of the matrix M as described, and we already know that later applications of Theorem B.0.16's procedure will leave $S(2^i + 1)$ unaffected, just as this application will leave unaffected $S(2^j + 1)$ for any $j < i$.

Theorem B.0.17's procedure does not affect S-box inputs less than 2^{h+1}, which exceeds $(2^i + 1)$.

We thus see how, by incorporating the procedures described in the proofs of Theorem B.0.19 and Lemma B.0.20 into our construction, we obtain an S-box with the desired properties. □

For an APN or D4U S-box, we can tighten the restrictions on $S(5)$ imposed by the above results further. We begin by ensuring that $S(5) \neq 7$. Consider the point in the procedure at which we would normally apply the methodology described in the proof of Lemma B.0.20:

Lemma B.0.23. *Let S be a bijective S-box for $n \geq 3$, such that S maps 0, 1, 2 and 4 to themselves, maps 3 to 5, and 5 to 7.*

Then there exists at least one S-box S_2 linear-equivalent to S, such that S_2 maps 0, 1, 2 and 4 to themselves, 3 to 5, and 5 to:

- *6, 9, 10, or 11. (if S is APN).*

- *3, 6, 9, 10, or 11. (otherwise).*

Proof. We currently have an S-box of the form

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 4 & 7 & X_1 & Y_1 & \dots \end{pmatrix}$$

for two unknown values (X_1, Y_1)

We apply a matrix identical to the identity matrix except that the last two rows of its last two columns are of the form

$$\begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}$$

to the inputs, and then to the outputs. (Or the outputs and then the inputs; the order we choose is irrelevant.)

The values with Hamming weight 1 will not map to themselves after the first matrix application; however they will again after the second, and the transformation will result in an S-box of the form:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 6 & 4 & X_2 & 7 & Y_2 & \dots \end{pmatrix}$$

Apply $CXOR(3, 11)$ so that 3 will map to 5. The S-box now takes the form:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 7 & X_3 & 4 & (Y_3 = Y_2 \text{ or } Y_2 \oplus 011) & \dots \end{pmatrix}$$

Operate on the inputs as described in the proof of Theorem B.0.17), mapping 4 to $S^{-1}(4) = 6$, so that 4 will again map to itself. We have:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 4 & Y_3 & 7 & X_3 & \dots \end{pmatrix}$$

Y_3 may be 6 (if so, Y_2 will have been 5 and Y_1 will have been 6), or it may be 3 if the S-box is not APN. If it is neither of these, we can apply the procedure described in the proof of Lemma B.0.20 to ensure that 5 maps to 9, 10 or 11 without changing the values mapped to by 0, 1, 2, 3 and 4, and as explained in the proof of Theorem B.0.22, this will remain the case after the full procedure has been applied to the S-box. □

If S is APN, we can then go on to ensure 5 will not map to 9:

Lemma B.0.24. *Let S be a bijective APN S-box for $n \geq 3$, such that S maps 0, 1, 2 and 4 to themselves, maps 3 to 5, and 5 to 6, 9, 10, or 11.*

Then there exists at least one S-box S_2 linear-equivalent to S, such that S_2 maps 0, 1, 2 and 4 to themselves, 3 to 5, and 5 to 6, 10, or 11.

Proof. If $S(5) \neq 9$, this is already true, and we do not need to do anything.

Otherwise, S is of the form

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 4 & 9 & Y_1 & Z_1 & \dots \end{pmatrix}$$

for two unknown values (Y_1, Z_1).

$S(6)$ and $S(7)$ cannot both be $\in \{10, 11\}$, since (input difference 1, output difference 1) already occurs for S-box inputs 0 and 1. It will therefore be possible to apply at least one of $CXOR(4, 011)$ or $CXOR(4, 010)$ so that 5 maps to either 10 or 11, without creating a situation where $S^{-1}(8) < 8$. As previously stated, we can now continue with the rest of the procedure without affecting the value of $S(5)$. □

However, prior to continuing with the rest of the procedure, we can eliminate 11 from the set of possible values for $S(5)$ if S is APN:

Lemma B.0.25. *Let S be a bijective APN S-box for $n \geq 3$, such that S maps 0, 1, 2 and 4 to themselves, maps 3 to 5, and 5 to 11.*

Then there exists at least one S-box S_2 linear-equivalent to S, such that S_2 maps 0, 1, 2 and 4 to themselves, 3 to 5, and 5 to 6 or 10.

Proof. S is of the form

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 4 & 11 & X_1 & Y_1 & \dots \end{pmatrix}$$

Y_1 cannot be equal to 9, otherwise we would have (input difference 2, output difference 2) for inputs 0, 2, 5, 7, and hence S would not be APN.

If X_1 is not equal to 9, we can simply apply $CXOR(4, 001)$ to the outputs.

If X_1 is equal to 9, we cannot do this, as otherwise it will result in the procedure of Result 14 having to be applied when ensuring $S(8) = 8$, undoing what we have achieved here. We have

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 4 & 11 & 9 & Y_1 & \dots \end{pmatrix}$$

(Note that $Y_1 \neq 8$ if S is APN.) Carry out the affine transformation which xors all inputs with 1, resulting in:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7\dots \\ 1 & 0 & 5 & 2 & 11 & 4 & Y_1 & 9\dots \end{pmatrix}$$

Carry out the same transformation on the outputs:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 4 & 3 & 10 & 5 & Y_2 & 8 & \dots \end{pmatrix}$$

Apply $CXOR(2, 1)$ to the inputs:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 3 & 4 & 10 & 5 & 8 & Y_2 & \dots \end{pmatrix}$$

Use the same $CXOR$ on the outputs:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 4 & 11 & 5 & 8 & Y_3 & \dots \end{pmatrix}$$

(Note that since S is APN, $Y_3 \neq 9$.)

Apply $CXOR(3, 01)$ to the outputs:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 11 & 4 & 8 & Y_4 & \dots \end{pmatrix}$$

(Since S is APN, $Y_4 \neq 9$.)

Apply the same $CXOR$ to the inputs:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 4 & 11 & Y_4 & 8 & \dots \end{pmatrix}$$

We reiterate that Y_4 cannot be equal to 9, or else we would have (input difference 1, output difference 1) for ((0, 0), (1, 1)) and ((6, 9), (7, 8)).

We now apply $CXOR(4, 001)$ to the outputs, obtaining:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ 0 & 1 & 2 & 5 & 4 & 10 & Y_5 & 9 & \dots \end{pmatrix}$$

Note that since Y_4 could not equal 9, Y_5 cannot equal 8. As a result, we will not have to apply the procedure of B.0.16, and the value of $S(5)$ will not be affected by any subsequent part of the overall procedure. □

From this, it follows that:

Corollary B.0.26. *Every bijective S-box S with differential uniformity ≤ 4 is affine-equivalent to at least one bijective S-box S_2 such that S_2 maps all inputs with Hamming weight less than 2 to themselves, 3 to 5, 5 to*

- *6 or 10 (if S is APN)*
- *3, 6, 9, 10, or 11 (otherwise)*

and all $2^i + 1 (3 \leq i \leq (n-1))$ to some value $\leq 2^{i+2} - 2i - 1$.

The question arises as to whether this representation is unique. We have tested this on various S-boxes by generating several affine-equivalent boxes and applying the procedure described above, and unfortunately the representation as described above (whether it be the more general result, the result assuming D4U, or the more restricted representation for APN S-boxes) has not in any of these cases been unique.

An additional result that applies for APN S-boxes.

Lemma B.0.27. *Let S be a bijective, APN S-box mapping all inputs with Hamming weight 1 to themselves. S does not map any inputs with Hamming weight 2 or 3 to themselves.*

Proof. For some value $a \in GF(2^n)$, let $HW(a)$ denote the Hamming weight of a. If some x with weight 2 was mapped to itself, input difference x, output difference x would occur for the two pairs $(0, x)$ and $(x, 0)$. However, it would also occur for $(2^k, 2^l)$ and $(2^l, 2^k)$ where $(2^k \oplus 2^l) = x$. This would imply that S had differential uniformity ≥ 4, contradicting our assertion that it was APN.

If some x with weight 3 was mapped to itself, let y be some input such that $HW(y) = 1$ and $HW(x \oplus y) = 2$. It would follow that $HW(S(x) \oplus S(y)) = 2$. Let z be the value $(x \oplus y)$. From z having weight 2, we see that (input difference z, output difference z) would

occur not only for the two input pairs $(x, y), (y, x)$, but also for $(z_1, z_2), (z_2, z_1)$ where z_1 and z_2 are the two values with Hamming weight 1 that would xor to give z. Again, this contradicts the assumption of almost-perfect nonlinearity. □

B.0.3 Consequences for genetic/memetic and ant algorithms

Genetic and memetic algorithms

The concept of *epistasis* in genetic algorithms is introduced in various tutorials on the subject [230, 223], and is also relevant in the context of memetic algorithms [102]. It is stated that, for such algorithms to be effective, the representation of the entities being evolved should be such that there is "little interaction between genes". In the context of the tutorial, the entities were strings of bits, and the individual genes were the individual bits. As far as possible, Townsend indicated, the effect of one bit's value on the fitness of the candidate solution should be independent of any other bit's effect.

While no rigorous mathematical definition based on the level of dependence was given, the representation of the candidate solutions was deemed to have high epistasis if the effects of certain bits on the fitness were in fact highly dependent on the values of other bits. This is not desirable.

In the case we are dealing with (bijections over $GF(2^n)$), the genes are the positions assigned to the individual output values. Clearly, if we use the unmodified truth table as the representation, the level of epistasis is extremely high. No individual output value contains any information, since given any S-box S with $S(x) = y$ and a given set of affine invariant properties, we may xor the set of outputs with any nonzero m-bit value to obtain S_2 with $S_2(x) \neq y$ and precisely the same set of properties. Even if we ensure $S(0) = 0$ to prevent this, using a different matrix to that defined in Theorem B.0.9 means that $(S(1), S(2))$ can still be mapped to any pair of arbitrary nonzero values. Clearly no information on the overall fitness exists until after we have fixed the values for these three truth table entries and at least one other - possibly not even then - and the effect of $S(x)$'s value for any $x > 2$ is clearly dependent on the values for these three and other entries.

We have attempted to counter this in our genetic and memetic algorithms by using the results of Section B.0.2 to fix the values of all $S(2^i)$ and $S(3)$, and to enforce the stated restrictions on the values of the $S(2^i + 1)$ so that these are consistent with the restrictions that would apply for an APN. Unfortunately, in experiments using the entries in the difference distribution table to calculate fitness, this has led to poorer average fitness

than when we have not done so, for the following reasons:

1. In the experiments, the mutation phase must either block mutations that would violate these restrictions, or allow them but then execute the full procedure of Section B.0.2 to render the candidate compliant again.

   - In the first case, this removed potential improvements from the set of mutations that would otherwise have been beneficial in terms of fitness.
   - In the second case, the number of changes in truth table entries resulting from the transformation procedure appears to have impaired the algorithm's convergence.

2. The impaired convergence may well result from the non-uniqueness of the representation described. As an example, we evolved an APN S-box for $n = 5$ and applied an algorithm that generated S-boxes affine-equivalent to it and then applied the procedure of Section B.0.2 to these until it was unable to find any more equivalent boxes satisfying these restrictions.

   In this experiment, the value of $S(6)$ took on every value $\in [17, 31] \backslash \{24\}$ in at least one of the thus-constructed S-boxes. $S(31)$ took on twenty different values, and the values in other positions were similarly varied. It would appear that the level of epistasis is still extremely high even after the affine transformations are used to fix and restrict truth table values as defined in Section B.0.2, and without knowledge of further transformations which could tighten the restrictions further, it is not clear how this can be addressed. Neither is there any alternate representation for affine equivalence group equivalence classes which would possess less epistasis, and all known alternate representations for bijections over $GF(2^n)$ are themselves in one-to-one correspondence with different truth tables.

3. The effects as described affected not only the mutation phase, but also the local optimisation phase when memetic algorithms were used. Since the local optimisation phase involved the equivalent of several mutations per individual at each generation, the loss of performance was even more marked.

In addition to the above, there exist more general notions of equivalence than affine-equivalence: EA-equivalence (Definition 5.1.15) and CCZ-equivalence (Definition 5.1.16).

Both of these generalise affine equivalence, and their invariant properties include the frequency with which each DDT entry occurs, as also the frequencies of absolute values of entries in the cryptographically relevant autocorrelation table and linear approximation table (the latter being particularly important in calculating the crucial nonlinearity property.) This fact almost certainly serves to increase the level of epistasis further, as non affine-equivalent S-boxes may still be CCZ or EA-equivalent, and possess identical fitness in terms of these tables. At this time, we do not know how to exploit the CCZ and EA transformations to impose further restrictions on output values that might counter this, and are forced to leave this as a matter for future research.

Finally, we note that experiments were made in using the restrictions in conjunction with simulated annealing, by preventing moves that would violate the constraints. The loss in performance observed was consistent with that observed for the memetic algorithms as noted.

Ant algorithms

In experiments for $n = 5$, the restrictions as described again led to worse performance for all parameter choices for the Ant System [120] and for two versions of Ant Colony System [119, 175]. Why this was so is not so clear as in the case of memetic algorithms, although local optimisation used in these algorithms would have been affected for the same reasons (either due to restricted moves, or retransformations impairing convergence.)

Appendix C

Errors in the description of the Dunkelman/Keller approximation

In the original description of Biham et al.'s linear approximation [27], on page 20, after S6 is applied the only active bit in the state is bit 30. In the descriptions given in later papers [31, 123], after the application of S6, bit 28 is shown as active instead of bit 30. In private email correspondence, one of the authors informed us that bit 28 was correct.

The linear diffusion layer is then applied, after which the active bits according to the diagram are 80, 101 and 103. However, by examining the description of Serpent's diffusion layer in its AES proposal [5], and the C reference implementation [7], we see that the xor of diffusion layer output bits {80, 101, 103} is the xor of input bits {4, 22, 35, 44, 46, 57, 62, 75, 86, 96, 97} - and is therefore unaffected by either bit 28 or bit 30. In the same correspondence mentioned above, this was revealed to be another typographical error - the active bits at this point should in fact have been 81, 83 and 100.

Bibliography

[1] Boost C++ libraries. `http://www.boost.org/`.

[2] A.M.B. Albassal and A-M.A. Wahdan. Genetic algorithm cryptanalysis of the basic substitution permutation network. In *Proceedings of the 2003 IEEE Midwest International Symposium on Circuits and Systems (MWSCAS '03)*. IEEE, December 2003. doi:10.1109/MWSCAS.2003.1562320.

[3] A.M.B. Albassal and A-M.A. Wahdan. Genetic algorithm cryptanalysis of a Feistel type block cipher. In *Proceedings of the 2004 International Conference on Electrical, Electronic and Computer Engineering (ICEEC '04)*, pages 217–221. IEEE, June 2004.

[4] A.M.B. Albassal and A-M.A. Wahdan. Neural network based cryptanalysis of a Feistel type block cipher. In *Proceedings of the 2004 International Conference on Electrical, Electronic and Computer Engineering (ICEEC '04)*, pages 231–237. IEEE, June 2004.

[5] R. Anderson, E. Biham, and L. Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. `http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf`.

[6] R. Anderson, E. Biham, L. Knudsen, and F. Stajano. Serpent optimised (bitslice) reference implementation. 1998. C source code. Can be downloaded from `http://www.cl.cam.ac.uk/~rja14/Papers/serpent.tar.gz`; is contained in the archive's "floppy2" folder.

[7] R. Anderson, E. Biham, L. Knudsen, and F. Stajano. Serpent reference implementation. 1998. C source code. Can be downloaded from `http://www.cl.cam.ac.uk/~rja14/Papers/serpent.tar.gz`; is contained in the archive's "floppy1" folder.

[8] A.G. Bafghi and B. Sadeghiyan. Finding suitable differential characteristics for block ciphers with ant colony technique. In *Proceedings of the Ninth IEEE Symposium on Computers and Communications, 2004. (ISCC 2004), volume 1 of 2*, pages 418–423. IEEE, 28 June - 1 July 2004. doi:10.1109/ISCC.2004.1358440.

[9] A.G. Bafghi, B. Sadeghiyan, and R. Safabakhsh. Finding the differential characteristics of block ciphers with neural networks. *Information Sciences*, 178(15):3118–3132, August 2008. doi:10.1016/j.ins.2008.02.016.

[10] T. Baignères, P. Junod, and S. Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *Advances in Cryptology - Asiacrypt 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 432–450. IACR, Springer, December 2004.

[11] H.J.C. Barbosa and C.C.H. Borges. A non-generational genetic algorithm for multiobjective optimization. In *Proceedings of the 2000 IEEE Congress on Evolutionary Computation (CEC2000), volume 1 of 2*, pages 172–179. IEEE, July 2000.

[12] H. Barnum, H. J. Bernstein, and L. Spector. Genetic programming for quantum computers. In *Proceedings of the 3rd Annual Conference on Genetic Programming (GP-98)*, pages 365–373. Morgan Kaufmann, July 1998.

[13] H. Barnum, H. J. Bernstein, and L. Spector. Quantum circuits for OR and AND of ORs. *Journal of Physics A: Mathematical and General*, 33(45):8047–8057, November 2000.

[14] H. Barnum, H. J. Bernstein, L. Spector, and N. Swamy. Finding a better-than-classical quantum AND/OR algorithm using genetic programming. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC1999)*. IEEE, July 1999.

[15] H. Barnum, H.J. Bernstein, L. Spector, and N. Swamy. *Quantum computing applications of genetic programming*, pages 135–160. The MIT Press, 1999. Published in "Advances in Genetic Programming", volume 3.

[16] E.B. Baum, D. Boneh, and C. Garrett. On genetic algorithms. In *Proceedings of the 8th Annual ACM conference on Computational Learning Theory, (COLT 1995)*, pages 230–239. ACM, 1995.

[17] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. *SOSEMANUK, a Fast Software-Oriented Stream Cipher*, volume 4986 of *Lecture Notes in Computer Science*, pages 98–118. Springer, 2008. Chapter 9 of "New Stream Cipher Designs - The eSTREAM Finalists".

[18] C. Berbain, O. Billet, A. Canteaut, B. Debraize, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. *$DECIM^{v2}$, a Fast Software-Oriented Stream Cipher*, volume 4986 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2008. Chapter 11 of "New Stream Cipher Designs - The eSTREAM Finalists".

[19] K.P. Bergmann. Cryptanalysis using nature-inspired optimization algorithms. Master's thesis, University of Calgary, August 2007. `https://dspace1.acs.ucalgary.ca/handle/1880/46384`.

[20] K.P. Bergmann, C. Jacob, and R. Scheidler. Cryptanalysis using genetic algorithms. In M. Keijzer, editor, *Proceedings of the 2008 Genetic and Evolutionary Computation COnference (GECCO 2008)*, pages 1099–1100. ACM, July 2008.

[21] D.J. Bernstein. Salsa20 design. `http://cr.yp.to/snuffle/design.pdf`.

[22] D.J. Bernstein. The Salsa20 family of stream ciphers. Available from author's website. December 2007. `http://cr.yp.to/snuffle/salsafamily-20071225.pdf`.

[23] L. Bianchi, L. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In J. Guervs, P. Adamidis, H-G. Beyer, H-P. Schwefel, and J-L. Fernndez-Villacaas, editors, *Parallel Problem Solving from Nature PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 883–892. Springer, 2002.

[24] E. Biham. A fast new DES implementation in software. In E. Biham, editor, *Proceedings of the Fourth International Workshop on Fast Software Encryption (FSE 1997)*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. IACR, Springer, January 1997.

[25] E. Biham, A. Biryukov, and A. Shamir. Miss in the middle attacks on IDEA and Khufu. In L.R. Knudsen, editor, *Proceedings of the Sixth International Workshop on*

Fast Software Encryption (FSE 1999), volume 1636 of *Lecture Notes in Computer Science*, pages 124–138. IACR, Springer, March 1999.

[26] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. *Journal of Cryptology*, 18(4):291–311, September 2005.

[27] E. Biham, O. Dunkelman, and N. Keller. Linear cryptanalysis of reduced round Serpent. In M. Matsui, editor, *Proceedings of the Eighth International Workshop on Fast Software Encryption (FSE 2001)*, volume 2355 of *Lecture Notes in Computer Science*, pages 16–27. IACR, Springer, April 2001.

[28] E. Biham, O. Dunkelman, and N. Keller. The rectangle attack - rectangling the Serpent. In B. Pfitzmann, editor, *Advances in Cryptology - Eurocrypt 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. IACR, Springer, 2001.

[29] E. Biham, O. Dunkelman, and N. Keller. Enhancing differential-linear cryptanalysis. In Y. Zheng, editor, *Advances in Cryptology - Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 587–592. IACR, Springer, 2002.

[30] E. Biham, O. Dunkelman, and N. Keller. New results on boomerang and rectangle attacks. In J. Daemen and V. Rijmen, editors, *Proceedings of the Ninth International Workshop on Fast Software Encryption (FSE 2002)*, volume 2365 of *Lecture Notes in Computer Science*, pages 1–16. IACR, Springer, February 2002.

[31] E. Biham, O. Dunkelman, and N. Keller. Differential-linear cryptanalysis of Serpent. In T. Johansson, editor, *Proceedings of the Tenth International Workshop on Fast Software Encryption (FSE 2003)*, volume 2887 of *Lecture Notes in Computer Science*, pages 9–21. IACR, Springer, February 2003.

[32] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. Technical Report CS90-16, Weizmann Institute of Science, July 1990. `http://www.cs.technion.ac.il/~biham/Reports/Weizmann/cs90-16.ps.gz`.

[33] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems (extended abstract). In A.J. Menezes and S.A. Vanstone, editors, *Advances in Cryptology - Crypto '90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. IACR, Springer, 1990.

[34] E. Biham and A. Shamir. Differential cryptanalysis of the full 16-round DES. In E.F. Brickell, editor, *Advances in Cryptology - Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. IACR, Springer, 1992.

[35] O. Billet and M. Robshaw, editors. *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*. Springer, 2008.

[36] A. Biryukov. Product Cipher, Superencryption. In H.C.A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*, page 480. Springer, 2005.

[37] A. Biryukov. Substitution-Permutation (SP) Network. In H.C.A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*, page 602. Springer, 2005.

[38] A. Biryukov, C. De Cannière, and M. Quisquater. On multiple linear approximations. In M. Franklin, editor, *Advances in Cryptology - Crypto 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 1–22. IACR, Springer, 2004.

[39] A. Biryukov and E. Kushilevitz. Improved cryptanalysis of RC5. In K. Nyberg, editor, *Advances in Cryptology - Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 85–99. IACR, Springer, May 1998.

[40] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In P. Paillier and I. Verbauwhede, editors, *Proceedings of the Ninth International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007)*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. IACR, Springer, September 2007.

[41] J. Borghoff, L.R. Knudsen, and M. Stolpe. Bivium as a mixed-integer linear programming problem. In M.G. Parker, editor, *Proceedings of the 12th IMA International Conference on Cryptography and Coding*, volume 5921 of *Lecture Notes in Computer Science*, pages 133–152. Springer, December 2009.

[42] N. Borisov, M. Chew, R. Johnson, and D. Wagner. Multiplicative differentials. In J. Daemen and V. Rijmen, editors, *Proceedings of the Ninth International Workshop on Fast Software Encryption (FSE 2002)*, volume 2365 of *Lecture Notes in Computer Science*, pages 17–33. IACR, Springer, February 2002.

[43] A. Braeken, J. Lano, and B. Preneel. Evaluating the resistance of stream ciphers with linear feedback against fast algebraic attacks. In L.M. Batten and R. Safavi-Naini, editors, *Proceedings of the Eleventh Australasian Conference on Information Security and Privacy (ACISP 2006)*, volume 4058 of *Lecture Notes in Computer Science*, pages 40–51. Springer, July 2006.

[44] S.L. Braunstein, I.D.K. Brown, S. Stepney, and A. Sudbery. Searching for highly entangled multi-qubit states. *Journal of Physics A: Mathematical and General*, 38(5):1119–1131, February 2005.

[45] K.A. Browning, J.F. Dillon, M.T. McQuistan, and A.J. Wolfe. An APN permutation in dimension six. In G. McGuire, G.L. Mullen, D. Panario, and I.E. Shparlinski, editors, *Finite Fields: Theory and Applications, Ninth International Conference*, volume 518 of *Contemporary Mathematics*, pages 33–42. AMS, July 2009.

[46] L. Budaghyan, C. Carlet, and A. Pott. New classes of almost bent and almost perfect nonlinear polynomials. *IEEE Transactions on Information Theory*, 52(3):1141–1152, March 2006.

[47] E.K. Burke, J.P. Newall, and R.F. Weare. A memetic algorithm for university exam timetabling. In E. Burke and P. Ross, editors, *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 241–250. Springer, August 29 - September 1 1995.

[48] L. Burnett, G. Carter, A. Clark, E. Dawson, and W. Millan. Evolutionary heuristics for finding cryptographically strong s-boxes. In Vijay Varadharajan and Yi Mu, editors, *Proceedings of the 2nd International Conference on Information and Communications Security (ICICS 1999)*, volume 1726 of *Lecture Notes in Computer Science*, pages 263–274. Springer, November 1999.

[49] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S.M. Matyas Jr, L. O'Connor, M. Peyravian, D. Safford, and N. Zunic. MARS - a candidate cipher for AES. Technical report, IBM, September 1999. `http://www.research.ibm.com/security/mars.pdf`.

[50] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt*

2000, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. IACR, Springer, May 2000.

[51] C. Carlet. Private communication.

[52] C. Carlet. Boolean functions for cryptography and error-correcting codes. In Y. Crama and P. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*. Cambridge University Press, 2010. The chapter is downloadable from `http://www.math.univ-paris13.fr/~carlet/chap-fcts-Bool-corr.pdf`.

[53] C. Carlet. Vectorial Boolean functions for cryptography. In Y. Crama and P. Hammer, editors, *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*. Cambridge University Press, 2010. The chapter is downloadable from `http://www.math.univ-paris13.fr/~carlet/chap-vectorial-fcts-corr.pdf`.

[54] C. Carlet. Comments on "Constructions of cryptographically significant Boolean functions using primitive polynomials". *IEEE Transactions on Information Theory*, 57(7):4852–4853, July 2011.

[55] C. Carlet and K. Feng. An infinite class of balanced functions with optimal algebraic immunity, good immunity to fast algebraic attacks and good nonlinearity. In J. Pieprzyk, editor, *Advances in Cryptology - Asiacrypt 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 425–440. IACR, Springer, December 2008.

[56] C. Carlet, L. Hu, J. Shan, and X. Zeng. More balanced Boolean functions with optimal algebraic immunity and good nonlinearity and resistance to fast algebraic attacks. *IEEE Transactions on Information Theory*, 57(9):6310–6320, September 2011.

[57] C. Carlet, W. Meier, and E. Pasalic. Algebraic attacks and decomposition of Boolean functions. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - Eurocrypt 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. IACR, Springer, May 2004.

[58] C. Carlet, D. Tang, and X. Tang. Highly nonlinear Boolean functions with optimal algebraic immunity and good behavior against fast algebraic attacks. Cryptology ePrint Archive, Report 2011/366. July 2011. `http://eprint.iacr.org/2011/366`.

[59] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, Jul-Dec 1998.

[60] G. Di Caro, M. Dorigo, and L. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[61] F. Chabaud and S. Vaudenay. Links between differential and linear cryptanalysis. In A. De Santis, editor, *Advances in Cryptology - Eurocrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 356–365. IACR, Springer, May 1994.

[62] H.W. Chan, K.P. Chow, C.F. Chong, L.C.K. Hui, W.W. Tsang, and X.Y. Wang. The differential cryptanalysis of an AES finalist - Serpent. Technical Report TR-2000-04, The University of Hong Kong, April 2000. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.1292`.

[63] P. Chardaire, J.L. Lutton, and A. Sutter. Thermostatistical persistency: A powerful improving concept for simulated annealing algorithms. *European Journal of Operational Research*, 86(3):565–579, November 1995.

[64] H. Chen, J.A. Clark, and J.L. Jacob. Automated design of security protocols. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC2003)*. IEEE, December 2003.

[65] Jung Hee Cheon and Dong Hoon Lee. Resistance of S-boxes against algebraic attacks. In B. Roy and W. Meier, editors, *Proceedings of the Eleventh International Workshop on Fast Software Encryption (FSE 2004)*, volume 3017 of *Lecture Notes in Computer Science*, pages 83–94. IACR, Springer, February 2004.

[66] Joo Yeon Cho. Linear cryptanalysis of reduced-round PRESENT. Cryptology ePrint Archive, Report 2009/397. 2009. `http://eprint.iacr.org/2009/397`.

[67] Joo Yeon Cho. Linear cryptanalysis of reduced-round PRESENT. In J. Pieprzyk, editor, *Proceedings of the Cryptographers' Track at the RSA Conference, 2010 (CT-RSA 2010)*, volume 5985 of *Lecture Notes in Computer Science*, pages 302–317. Springer, March 2010.

[68] J.Y. Cho, M. Hermelin, and K. Nyberg. Multidimensional linear cryptanalysis of reduced round Serpent. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors,

Proceedings of the Thirteenth Australasian Conference on Information Security and Privacy (ACISP 2008), volume 5107 of *Lecture Notes in Computer Science*, pages 203–215. Springer, July 2008.

[69] J.Y. Cho, M. Hermelin, and K. Nyberg. A new technique for multidimensional linear cryptanalysis with applications on reduced round Serpent. In Pil Joong Lee and Jung Hee Cheon, editors, *Proceedings of the 11th International Conference on Information Security and Cryptology (ICISC 2008)*, volume 5461 of *Lecture Notes in Computer Science*, pages 383–398. Springer, December 2008.

[70] J.Y. Cho, M. Hermelin, and K. Nyberg. Multidimensional extension of Matsui's algorithm 2. In O. Dunkelman, editor, *Proceedings of the Sixteenth International Workshop on Fast Software Encryption (FSE 2009)*, volume 5665 of *Lecture Notes in Computer Science*, pages 209–227. IACR, Springer, February 2009.

[71] J.Y. Cho, M. Hermelin, and K. Nyberg. Statistical tests for key recovery using multidimensional extension of Matsui's algorithm 1. In: Eurocrypt 2009 - poster session. April 2009. `http://research.ics.tkk.fi/publications/mhermeli/dags-unif-alg1.pdf`.

[72] A. Clark, H. Bergen, and E. Dawson. Combinatorial optimization and the knapsack cipher. *Cryptologia*, 20(1):85–93, January 1996. doi:10.1080/0161-119691884807.

[73] J.A. Clark. My personal hotlist. Page of University of York's Computer Science department website. `http://web.archive.org/web/20081201143929/http://www.cs.york.ac.uk/security/LibraryPages/NatureInspired.html`.

[74] J.A. Clark. *Metaheuristic Search as a Cryptological Tool*. PhD thesis, University of York, December 2001.

[75] J.A. Clark. Lecture notes from University of York "Cryptography, Attacks and Countermeasures" (CRY) module. 2009.

[76] J.A. Clark, J.C Hernández-Castro, and J.M.E. Tapiador. Private communication. C source code used in "Non-linear Cryptanalysis Revisited: Heuristic Search for Approximations to S-Boxes".

[77] J.A. Clark, J.C. Hernández-Castro, and J.M.E. Tapiador. Heuristic search for non-linear cryptanalytic approximations. In *Proceedings of the 2007 IEEE Congress on*

Evolutionary Computation (CEC2007), pages 3561–3568. IEEE, September 2007. doi:10.1109/CEC.2007.4424934.

[78] J.A. Clark, J.C Hernández-Castro, and J.M.E. Tapiador. Non-linear cryptanalysis revisited: Heuristic search for approximations to S-boxes. In S.D. Galbraith, editor, *Proceedings of the 11th IMA International Conference on Cryptography and Coding*, volume 4887 of *Lecture Notes in Computer Science*, pages 99–117. Springer, December 2007.

[79] J.A. Clark and J. Jacob. Fault injection and a timing channel on an analysis technique. In *Advances in Cryptology - Eurocrypt 2002*, volume 2332 of *Lecture Notes in Computer Science*. IACR, Springer, April 28 - May 2 2002.

[80] J.A. Clark, J. Jacob, S. Stepney, S. Maitra, and W. Millan. Evolving Boolean functions satisfying multiple criteria. In A. Menezes and P. Sarkar, editors, *Progress in Cryptology - Indocrypt 2002*, volume 2551 of *Lecture Notes in Computer Science*, pages 246–259. Springer, December 2002.

[81] J.A. Clark and J.L. Jacob. Searching for a solution: Engineering tradeoffs and the evolution of provably secure protocols. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy, (SP '00)*. IEEE, September 2000.

[82] J.A. Clark and J.L. Jacob. Two-stage optimisation in the design of Boolean functions. In C. Boyd, A. Clark, and A. Dawson, editors, *Proceedings of the Fifth Australasian Conference on Information Security and Privacy (ACISP 2000)*, volume 1841 of *Lecture Notes in Computer Science*, pages 242–254. Springer, July 2000.

[83] J.A. Clark and J.L. Jacob. Protocols are programs too: the meta-heuristic search for security protocols. *Information and Software Technology*, 43(14):891–904, December 2001.

[84] J.A. Clark, J.L. Jacob, S. Maitra, and P. Stanica. Almost Boolean functions: The design of Boolean functions by spectral inversion. *Computational Intelligence*, 20(3):450–462, August 2004.

[85] J.A. Clark, J.L. Jacob, and S. Stepney. Secret agents leave big footprints: How to plant a cryptographic trapdoor, and why you might not get away with it. In *Proceedings of GECCO (Genetic and Evolutionary Computation COnference) 2003,*

part 2, volume 2724 of *Lecture Notes in Computer Science*, pages 2022–2033. International Society for Genetic and Evolutionary Computation (ISGEC) (former name; now ACM SIGEVO), Springer, July 2003.

[86] J.A. Clark, J.L. Jacob, and S. Stepney. The design of S-boxes by simulated annealing. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC2004)*, pages 1533–1537. IEEE, June 2004. Volume 2.

[87] J.A. Clark, J.L. Jacob, and S. Stepney. Searching for cost functions. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC2004)*, pages 1517–1524. IEEE, June 2004. Volume 2.

[88] J.A. Clark, J.L. Jacob, and S. Stepney. The design of S-boxes by simulated annealing. *New Generation Computing*, 23(3):219–231, September 2005.

[89] J.A. Clark, S. Maitra, and P. Stanica. Results on rotation symmetric bent and correlation immune Boolean functions. In B. Roy and W. Meier, editors, *Proceedings of the Eleventh International Workshop on Fast Software Encryption (FSE 2004)*, volume 3017 of *Lecture Notes in Computer Science*, pages 161–177. IACR, Springer, February 2004.

[90] J.A. Clark, P. Massey, and S. Stepney. Evolving quantum circuits and programs through genetic programming. In *Proceedings of the 2004 Genetic and Evolutionary Computation COnference (GECCO 2004), Part 2 of 2*, pages 569–580. ACM, June 2004.

[91] J.A. Clark, P. Massey, and S. Stepney. Evolution of a human-competitive quantum Fourier transform algorithm using genetic programming. In H-G. Beyer and U-M O'Reilly, editors, *Proceedings of the 2005 Genetic and Evolutionary Computation COnference (GECCO 2005)*, pages 1657–1663. ACM, June 2005.

[92] J.A. Clark, P. Massey, and S. Stepney. Human-competitive evolution of quantum computing artefacts by genetic programming. *Evolutionary Computation*, 14(1):21–40, 2006.

[93] J.A. Clark, M. Russell, and S. Stepney. Making the most of two heuristics: breaking transposition ciphers with ants. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC2003)*. IEEE, December 2003.

[94] J.A. Clark, M. Russell, and S. Stepney. Using ants to attack a classical cipher. In *Proceedings of GECCO (Genetic and Evolutionary Computation COnference) 2003, part 1*, volume 2723 of *Lecture Notes in Computer Science*, pages 146–147. International Society for Genetic and Evolutionary Computation (ISGEC) (former name; now ACM SIGEVO), Springer, July 2003.

[95] J.A. Clark and S. Stepney. Evolving quantum programs and protocols. Published in the Handbook of Theoretical and Computational Nanotechnology (editors: Michael Rieth, Wolfram Schommers). 2005.

[96] J.A. Clark and S. Stepney. Searching for quantum programs and quantum protocols: a review. *Journal of Computational and Theoretical Nanoscience*, 5(5):942–969, May 2008. Pre-publication version can be found online at `http://www-users.cs.york.ac.uk/~jac/PublishedPapers/SearchingForQPsjctn-08.pdf`. This survey updates and revises "Evolving quantum programs and protocols" by the same authors.

[97] B. Collard and F.-X. Standaert. Experimenting linear cryptanalysis. 2011. `http://perso.uclouvain.be/fstandae/PUBLIS/90.pdf`.

[98] B. Collard, F.-X. Standaert, and J.-J. Quisquater. Improved and multiple linear cryptanalysis of reduced round Serpent. In Dingyi Pei, Moti Yung, Dongdai Lin, and Chuankun Wu, editors, *Proceedings of the 3rd SKLOIS Conference on Information Security and Cryptology (Inscrypt 2007)*, volume 4990 of *Lecture Notes in Computer Science*, pages 383–398. Springer, August 31 - September 5 2007.

[99] B. Collard, F.-X. Standaert, and J.-J. Quisquater. Improved and multiple linear cryptanalysis of reduced round Serpent - description of the linear approximations. 2007. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.522&rep=rep1&type=pdf`.

[100] B. Collard, F.-X. Standaert, and J.-J. Quisquater. Improving the time complexity of Matsui's linear cryptanalysis. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Proceedings of the 10th International Conference on Information Security and Cryptology (ICISC 2007)*, volume 4817 of *Lecture Notes in Computer Science*, pages 77–88. Springer, November 2007.

[101] D. Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38(3):243–250, May 1994.

[102] C. Cotta and F. Neri. *A Primer on Memetic Algorithms*, pages 43–52. Springer, 2012. Chapter 4 of "Handbook of Memetic Algorithms".

[103] N.T. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. IACR, Springer, August 2003.

[104] N.T. Courtois. Feistel schemes and bi-linear cryptanalysis (extended abstract). In M. Franklin, editor, *Advances in Cryptology - Crypto 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 23–40. IACR, Springer, August 2004.

[105] N.T. Courtois. Feistel schemes and bi-linear cryptanalysis. Cryptology ePrint Archive, Report 2005/251. August 2005. `http://eprint.iacr.org/2005/251`.

[106] N.T. Courtois. How fast can be algebraic attacks on block ciphers? Cryptology ePrint Archive, Report 2006/168. May 2006. `http://eprint.iacr.org/2006/168`.

[107] N.T. Courtois. CTC2 and fast algebraic attacks on block ciphers revisited. Cryptology ePrint Archive, Report 2007/152. May 2007. `http://eprint.iacr.org/2007/152`.

[108] N.T. Courtois, D. Hulme, and T. Mourouzis. Solving circuit optimisation problems in cryptography and cryptanalysis. Cryptology ePrint Archive, Report 2011/475. September 2011. `http://eprint.iacr.org/2011/475`.

[109] N.T. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In E. Biham, editor, *Advances in Cryptology - Eurocrypt 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. IACR, Springer, May 2003.

[110] N.T. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. Cryptology ePrint Archive, Report 2002/044. November 2002. `http://eprint.iacr.org/2002/044`.

[111] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley-Interscience, second edition, 2006.

[112] J. Daemen. *Cipher and Hash Function Design Strategies based on Linear and Differential Cryptanalysis.* PhD thesis, K.U.Leuven, March 1995.

[113] J. Daemen and V. Rijmen. AES submission document on Rijndael, Version 2 (with additional note on naming appended to start). September 1999. Note on naming added April 2003. `http://cs-www.ncsl.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf`.

[114] Zhenli Dai, Meiqin Wang, and Yue Sun. Effect of the dependent paths in linear hull. Cryptology ePrint Archive, Report 2010/325. 2010. `http://eprint.iacr.org/2010/325`.

[115] H.A. David. *Order Statistics.* Wiley, second edition, 1981.

[116] R. Dawkins. *The Selfish Gene.* Oxford University Press, 30th anniversary edition edition, 2006.

[117] E. Dawson, M. Henricksen, and L. Simpson. *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, chapter 3, pages 20–38. Springer, 2008. The relevant chapter is entitled "The Dragon Stream Cipher: Design, Analysis, and Implementation Issues".

[118] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[119] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[120] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1):29–41, February 1996.

[121] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. Technical Report IRIDIA-2000-32, Université Libre de Bruxelles, 2000. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.9732&rep=rep1&type=pdf`.

[122] O. Dunkelman. Private communication.

[123] O. Dunkelman, S. Indesteege, and N. Keller. A differential-linear attack on 12-round Serpent. In D.R. Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology - Indocrypt 2008*, volume 5365 of *Lecture Notes in Computer Science*, pages 308–321. Springer, December 2008.

[124] M. Fischer. DES how-to, version 1.2. Posting to sci.crypt. February 1995. `http://groups.google.co.uk/group/sci.crypt/msg/1d4d4f853629e3af`.

[125] S. Fischer and W. Meier. Algebraic immunity of s-boxes and augmented functions. In A. Biryukov, editor, *Proceedings of the Fourteenth International Workshop on Fast Software Encryption (FSE 2007)*, volume 4593 of *Lecture Notes in Computer Science*, pages 366–381. IACR, Springer, March 2007.

[126] R. Forré. A fast correlation attack on nonlinearly feedforward filtered shift-register sequences. In J-J Quisquater and J. Vandewalle, editors, *Advances in Cryptology - Eurocrypt '89*, volume 434 of *Lecture Notes in Computer Science*, pages 586–595. IACR, Springer, April 1989.

[127] J. Fuller and W. Millan. On linear redundancy in the AES S-box. Cryptology ePrint Archive, Report 2002/111. August 2002. `http://eprint.iacr.org/2008/385`. Full version published as "Linear Redundancy in S-Boxes" in proceedings of FSE 2008 (LNCS 2887).

[128] C. Gagné, M. Gravel, and W.L. Price. Scheduling continuous casting of aluminum using a multiple objective ant colony optimization metaheuristic. *European Journal of Operational Research*, 143(1):218–229, November 2002.

[129] J.J. Gillogly and T. Mahon. *Decoding the IRA*. Mercier Press, 2008.

[130] C. Harpes, G.G. Kramer, and J.L. Massey. A generalization of linear cryptanalysis and the applicability of Matsui's piling-up lemma. In L.C. Guillou and J-J. Quisqater, editors, *Advances in Cryptology - Eurocrypt '95*, volume 921 of *Lecture Notes in Computer Science*, pages 24–38. IACR, Springer, 1995.

[131] M. Hell, T. Johansson, A. Maximov, and W. Meier. *The Grain Family of Stream Ciphers*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2008. Chapter 14 of "New Stream Cipher Designs - The eSTREAM Finalists".

[132] T. Helleseth and S. RØnjom. A new attack on the filter generator. *IEEE Transactions on Information Theory*, 53(5):1752–1758, May 2007.

[133] M. Hellman and S.K. Langford. Differential-linear cryptanalysis. In Y. Desmedt, editor, *Advances in Cryptology - Crypto '94*, volume 839 of *Lecture Notes in Computer Science*, pages 17–25. IACR, Springer, 1994.

[134] M. Hellman and S. Pohlig. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, January 1978.

[135] M. Hermelin. *Multidimensional Linear Cryptanalysis*. PhD thesis, Aalto University, June 2010. `https://aaltodoc.aalto.fi/handle/123456789/4802`.

[136] M. Hermelin and K. Nyberg. Dependent linear approximations: The algorithm of Biryukov and others revisited. In J. Pieprzyk, editor, *Proceedings of the Cryptographers' Track at the RSA Conference, 2010 (CT-RSA 2010)*, volume 5985 of *Lecture Notes in Computer Science*, pages 318–333. Springer, March 2010.

[137] M. Hermelin and K. Nyberg. Linear cryptanalysis using multiple linear approximations. Cryptology ePrint Archive, Report 2011/093. February 2011. `http://eprint.iacr.org/2011/093`.

[138] J.C. Hernández-Castro and P. Isasi. Finding efficient distinguishers for cryptographic mappings, with an application to the block cipher TEA. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC2003)*, pages 2189–2193. IEEE, December 2003.

[139] J.C. Hernández-Castro, P. Isasi, A. Ribagorda, and J.M. Sierra. Genetic cryptoanalysis of two rounds TEA. In P.M.A. Sloot, co-editors J.J. Dongarra, A.G. Hoekstra, and C.J.K. Tan, editors, *Proceedings of the International Conference on Computational Science 2002 (ICCS 2002)*, volume 2331 of *Lecture Notes in Computer Science*, pages 1024–1031. Springer, April 2002.

[140] H.M. Heys. A tutorial on linear and differential cryptanalysis. Technical Report CORR 2001-17, University of Waterloo, March 2001. Available online, with errata, at `http://www.engr.mun.ca/~howard/Research/Papers/index.html`.

[141] H. Imai, X-M. Zhang, and Y. Zheng. Relating differential distribution tables to other properties of substitution boxes. *Designs, Codes and Cryptography*, 19(1):45–63, January 2000.

[142] P. Itaim and M.C. Riff. Applying differential cryptanalysis for XTEA using a genetic algorithm. Previously available from Riff's homepage (`http://www.inf.utfsm.cl/~mcriff/comet/publications/papers/alio-itaim-2008.pdf`), now no longer online. 2008.

[143] T. Johansson and Q. Wang. A note on fast algebraic attacks and higher order nonlinearities. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 6th International Conference (Inscrypt 2010)*, volume 6584 of *Lecture Notes in Computer Science*, pages 404–414. Springer, October 2010.

[144] B.S. Kaliski Jr. and M.J.B. Robshaw. Linear cryptanalysis using multiple approximations. In Y.G. Desmedt, editor, *Advances in Cryptology - Crypto '94*, volume 839 of *Lecture Notes in Computer Science*, pages 26–39. IACR, Springer, 1994.

[145] J. Nakahara Jr., P. Sepehrdad, M. Wang, and B. Zhang. Linear (hull) and algebraic cryptanalysis of the block cipher PRESENT. In J.A. Garay, A. Miyaji, and A. Otsuka, editors, *Proceedings of the 8th International Conference on Cryptology and Network Security (CANS 2009)*, volume 5888 of *Lecture Notes in Computer Science*, pages 58–75. Springer, December 2009.

[146] P. Junod. On the complexity of Matsui's attack. In S. Vaudenay and A.M. Youssef, editors, *Proceedings of the 8th Annual International Workshop on Selected Areas in Cryptography (SAC 2001)*, volume 2259 of *Lecture Notes in Computer Science*, pages 199–211. Springer, August 2001.

[147] D. Kahn. *The Codebreakers*. Sphere Books, abridged paperback edition, 1973.

[148] H. Kan, J. Peng, Q. Wang, and X. Xue. Constructions of cryptographically significant Boolean functions using primitive polynomials. *IEEE Transactions on Information Theory*, 56(6):3048–3053, June 2010.

[149] E. Kanter, I. Kanter, and W. Kinzel. Secure exchange of information by synchronization of neural networks. *Europhysics Letters*, 57(1):141–147, January 2002.

[150] J. Kelsey, T. Kohno, and B. Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In B. Schneier, editor, *Proceedings of the Seventh International Workshop on Fast Software Encryption (FSE 2000)*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. IACR, Springer, April 2000.

[151] J. Kelsey, T. Kohno, and B. Schneier. Preliminary cryptanalysis of reduced-round Serpent. In *Proceedings of the 3rd Advanced Encryption Standard Candidate Conference (AES 2000)*, pages 195–211, April 2000.

[152] J. Kelsey, B. Schneier, and D. Wagner. Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In N. Koblitz, editor, *Advances in Cryptology - Crypto '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. IACR, Springer, August 1996.

[153] J. Kelsey, B. Schneier, and D. Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *Proceedings of the 1st International Conference on Information and Communications Security (ICICS 1997)*, volume 1334 of *Lecture Notes in Computer Science*, pages 233–246. Springer, November 1997.

[154] E.F. Khor, T.H. Lee, and K.C. Tan. *Multiobjective Evolutionary Algorithms and Applications*. Springer, 2005.

[155] D. Khovratovich, G. Leurent, and C. Rechberger. Narrow-bicliques: Cryptanalysis of full IDEA. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - Eurocrypt 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 392–410. IACR, Springer, April 2012.

[156] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5-6):975–986, March 1984.

[157] S. Kirkpatrick and M.P. Vecchi. Global wiring by simulated annealing. *IEEE Transactions on Computer-Aided Design*, CAD-2(4):215–222, October 1983.

[158] A. Klimov, A. Mityagin, and A. Shamir. Analysis of neural cryptography. In Y. Zheng, editor, *Advances in Cryptology - Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 288–298. IACR, Springer, December 2002.

[159] L.R. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *Proceedings of the Second International Workshop on Fast Software Encryption (FSE 1994)*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. IACR, Springer, December 1994.

[160] L.R. Knudsen. DEAL - a 128-bit block cipher. Hosted on CiteSeerX. February, revised May 1998. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.7982`.

[161] L.R. Knudsen. October 2008. Private communication.

[162] L.R. Knudsen, G. Leander, A. Poschmann, and M.J.B. Robshaw. PRINTCIPHER: A block cipher for IC-printing. In S. Mangard and F-X. Standaert, editors, *Proceedings of the Twelfth International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2010)*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. IACR, Springer, August 2010.

[163] L.R. Knudsen and W. Meier. Cryptanalysis of an identification scheme based on the Permuted Perceptron Problem. In J. Stern, editor, *Advances in Cryptology - Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 363–374. IACR, Springer, May 1999.

[164] L.R. Knudsen and M.J.B. Robshaw. Non-linear approximations in linear cryptanalysis. In U. Maurer, editor, *Advances in Cryptology - Eurocrypt '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 224–236. IACR, Springer, 1996.

[165] J. Kolodziejczyk. The application of genetic algorithm in cryptoanalysis of knapsack cipher. Presented at Eurogen '97 - A short course on Genetic Algorithms and Evolution Strategies in computational science and engineering. 1997. `http://ceani.ulpgc.es/ingenetcd/eurogenxx/eurogen97/contributed/kolodziejczyk/ht/kolodziejczyk.htm`.

[166] H.O. Kunz. On the equivalence between one-dimensional discrete Walsh-Hadamard and multidimensional discrete Fourier transforms. *IEEE Transactions on Computers*, C-28(3):267–268, March 1979.

[167] M. Kwan. Reducing the gate count of bitslice DES. Cryptology ePrint Archive, Report 2000/051. October 2000. `http://eprint.iacr.org/2000/051`.

[168] M. Kwan. Post-postscript. Kwan's homepage, http://www.darkside.com.au/bitslice/. June 2011.

[169] X. Lai. Higher order derivatives and differential cryptanalysis. In R.E. Blahut, D.J. Costello Jr, U. Maurer, and T. Mittelholzer, editors, *Communications and Cryptography - Two Sides of One Tapestry*, pages 227–233. Kluwer Academic Publishers, 1994. Scanned copy online at http://cr.yp.to/cubeattacks.html.

[170] X. Lai and J.L. Massey. A proposal for a new block encryption standard. In I.B. Damgård, editor, *Advances in Cryptology - Eurocrypt '90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. IACR, Springer, May 1990.

[171] G. Leander. Small scale variants of the block cipher PRESENT. Cryptology ePrint Archive, Report 2010/143. March 2010. http://eprint.iacr.org/2010/143.

[172] G. Leander. On linear hulls, statistical saturation attacks, PRESENT and a cryptanalysis of PUFFIN. In K.G. Paterson, editor, *Advances in Cryptology - Eurocrypt 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 303–322. IACR, Springer, May 2011.

[173] G. Leander and A. Poschmann. On the classification of 4 bit s-boxes. In Claude Carlet and Berk Sunar, editors, *Proceedings of the First International Workshop on Arithmetic of Finite Fields (WAIFI 2007)*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, June 2007.

[174] A.K. Lenstra, H.W. Lenstra Jr., and L.Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982.

[175] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2009. Available at http://cs.gmu.edu/~sean/book/metaheuristics/.

[176] A.W. Machado. The Nimbus cipher. October 2000. https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/nimbus.zip.

[177] P.S. Massey. *Searching for Quantum Software*. PhD thesis, University of York, 2006. http://www.cs.york.ac.uk/ftpdir/reports/2007/YCST/11/YCST-2007-11.pdf.

[178] K. Mathias, S. McDaniel, T. Starkweather, C. Whitley, and D. Whitley. A comparison of genetic sequencing operators. In R.K. Belew and L.B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA 1991)*, pages 69–76. Morgan Kaufmann, July 1991.

[179] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology - Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. IACR, Springer, 1993.

[180] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Y.G. Desmedt, editor, *Advances in Cryptology - Crypto '94*, volume 839 of *Lecture Notes in Computer Science*, pages 1–11. IACR, Springer, 1994.

[181] Q. Meng, J. Song, Z. Wang, and H. Zhang. Cryptanalysis of four-round DES based on genetic algorithm. In *Proceedings of the 2007 International Conference on Wireless Communications, Networking and Mobile Computing (WiCom 2007)*. IEEE, September 2007. doi:10.1109/WICOM.2007.580.

[182] J. Monnerat and S. Vaudenay. Generic homomorphic undeniable signatures - erratum. February 2005. `http://infoscience.epfl.ch/record/99428/files/MV04d_note.pdf`.

[183] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. Technical report, California Institute of Technology, 1989. `http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.9474`.

[184] N. Mouha, B. Preneel, Y. Sun, and M. Wang. Algebraic techniques in differential cryptanalysis revisited. In Udaya Parampalli and Philip Hawkes, editors, *Proceedings of the Sixteenth Australasian Conference on Information Security and Privacy (ACISP 2011)*, volume 6812 of *Lecture Notes in Computer Science*, pages 120–141. Springer, July 2011.

[185] D. M'Raïhi, D. Naccache, J. Stern, and S. Vaudenay. XMX: A firmware-oriented block cipher based on modular multiplications. In E. Biham, editor, *Proceedings of the Fourth International Workshop on Fast Software Encryption (FSE 1997)*, volume 1267 of *Lecture Notes in Computer Science*, pages 166–171. IACR, Springer, January 1997. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.3608`.

[186] S. Murphy. The independence of linear approximations in symmetric cryptanalysis. *IEEE Transactions on Information Theory*, 52(12):5510–5518, December 2006. http://www.isg.rhul.ac.uk/~sean/Mult_lin_chi_IEEE_FinalA.pdf.

[187] S. Murphy. The effectiveness of the linear hull effect. Technical Report RHUL-MA-2009-19, Royal Holloway, University of London, October 2009. http://www.isg.rhul.ac.uk/~sean/Linear_Hull_JMC-Rev2-llncs.pdf.

[188] S. Murphy. Overestimates for the gain of multiple linear approximations. Technical Report RHUL-MA-2009-21, Royal Holloway, University of London, October 2009. http://www.isg.rhul.ac.uk/~sean/MLA_Gain.pdf.

[189] S. Murphy. Overestimates for the gain of multiple linear approximations in symmetric cryptology. *IEEE Transactions on Information Theory*, 57(7):4794–4797, July 2011.

[190] N. Nalini and G. Raghavendra Rao. Attacks of simple block ciphers via efficient heuristics. *Information Sciences*, 177(2):2553–2569, June 2007.

[191] National Institute for Science and Technology (NIST). Data Encryption Standard (DES). October 1999. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[192] National Institute for Science and Technology (NIST). Advanced Encryption Standard (FIPS PUB 197). November 2001. http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[193] R.M. Needham and D.J. Wheeler. TEA, a Tiny Encryption Algorithm. In B. Preneel, editor, *Proceedings of the Second International Workshop on Fast Software Encryption (FSE 1994)*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. IACR, Springer, December 1994.

[194] R.M. Needham and D.J. Wheeler. Tea extensions. Technical report, University of Cambridge, October 1997. http://www.cix.co.uk/~klockstone/xtea.pdf.

[195] R.M. Needham and D.J. Wheeler. Correction to xtea. Technical report, University of Cambridge, October 1998. http://www.movable-type.co.uk/scripts/xxtea.pdf.

[196] Phuong Ha Nguyen, Hongjun Wu, and Huaxiong Wang. Improving the algorithm 2 in multidimensional linear cryptanalysis. In Udaya Parampalli and Philip Hawkes, editors, *Proceedings of the Sixteenth Australasian Conference on Information Security and Privacy (ACISP 2011)*, volume 6812 of *Lecture Notes in Computer Science*, pages 61–74. Springer, July 2011.

[197] K. Nyberg. Differentially uniform mappings for cryptography. In T. Helleseth, editor, *Advances in Cryptology - Eurocrypt '93*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. IACR, Springer, 1993.

[198] K. Nyberg. Linear approximation of block ciphers. In A. De Santis, editor, *Advances in Cryptology - Eurocrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 439–444. IACR, Springer, May 1994.

[199] K. Nyberg. Multidimensional extension of Matsui's algorithm 2 - a revision of subsection 5.1. Private email communication, 2011.

[200] K. Nyberg and A. Röck. Exploiting linear hull in Matsui's algorithm 1 (extended version). Cryptology ePrint Archive, Report 2011/285. May 2011. `http://eprint.iacr.org/2011/285`.

[201] K. Ohkuma. Weak keys of reduced-round PRESENT for linear cryptanalysis. In M.J. Jacobson Jr, V. Rijmen, and R. Safavi-Naini, editors, *Proceedings of the 16th Annual International Workshop on Selected Areas in Cryptography (SAC 2009)*, volume 5867 of *Lecture Notes in Computer Science*, pages 249–265. Springer, August 2009.

[202] D.A. Osvik. Speeding up Serpent. In *Proceedings of the 3rd Advanced Encryption Standard Candidate Conference (AES 2000)*, April 2000.

[203] E. Pasalic. On cryptographically significant mappings over $GF(2^n)$. In Joachim von zur Gathen, José Luis Imaña, and Çetin Kaya Koç, editors, *Proceedings of the Second International Workshop on Arithmetic of Finite Fields (WAIFI 2008)*, volume 5130 of *Lecture Notes in Computer Science*, pages 189–204. Springer, July 2008.

[204] D. Pointcheval. Neural networks and their cryptographic applications. In P. Charpin, editor, *Livre des résumés EUROCODE '94*, pages 183–193, October 1994.

[205] D. Pointcheval. A new identification scheme based on the Perceptrons Problem. In L.C. Guillou and J-J. Quisqater, editors, *Advances in Cryptology - Eurocrypt '95*, volume 921 of *Lecture Notes in Computer Science*, pages 319–328. IACR, Springer, May 1995.

[206] D. Pointcheval and G. Poupard. A new NP-complete problem and public-key identification. *Designs, Codes and Cryptography*, 28(1):5–31, January 2003.

[207] B. Preneel, B. Van Rompay, S. B. Ors, A. Biryukov, L. Granboulan, E. Dottax, M. Dichtl, M. Schafheutle, P. Serf, S. Pyka, E. Biham, E. Barkan, O. Dunkelman, J. Stolin, M. Ciet, J-J. Quisquater, F. Sica, H. Raddum, and M. Parker. Performance of optimized implementations of the NESSIE primitives (version 2.0). February 2003. `http://www.cosic.esat.kuleuven.be/nessie/deliverables/D21-v2.pdf`.

[208] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comunications of the ACM*, 21(2):120–126, February 1978.

[209] F. Rubin. Comments on "Cryptanalysis of Knapsack Ciphers Using Genetic Algorithms". *Cryptologia*, 18(2):153–154, April 1994. doi:10.1080/0161-119491882838.

[210] H.V. Sahasrabuddhe and I.F.T. Yaseen. A genetic algorithm for the cryptanalysis of Chor-Rivest knapsack public key cryptosystem (PKC). In *Proceedings of the Third International Conference on Computational Intelligence and Multimedia Applications, 1999. (ICCIMA '99)*, pages 81–85. IEEE, September 1999. doi:10.1109/ICCIMA.1999.798506.

[211] B. Schneier. *Applied Cryptography*. Wiley, second edition, 1996.

[212] Beale Screamer. Microsoft's digital rights management scheme - technical details. October 2001. `http://cryptome.org/beale-sci-crypt.htm`.

[213] A.A. Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21(1):131–147, January 2008.

[214] A. Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE Transactions on Information Theory*, 30(5):699–704, September 1984.

[215] C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[216] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, 30(5):776–780, September 1984.

[217] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, C-34(1):81–85, January 1985.

[218] S. Singh. *The Code Book: The Secret History of Codes and Codebreaking*. Fourth Estate, 1999.

[219] L. Spector. *Automatic Quantum Computer Programming: A Genetic Programming Approach*. Kluwer Academic Publishers, first edition, 2004.

[220] R. Spillman. Cryptanalysis of knapsack ciphers using genetic algorithms. *Cryptologia*, 17(4):367–377, October 1993. doi:10.1080/0161-119391867999.

[221] O. Staffelbach and W. Meier. Fast correlation attacks on stream ciphers (extended abstract). In C.G. Günther, editor, *Advances in Cryptology - Eurocrypt '88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. IACR, Springer, May 1988.

[222] G. Sullivan and F. Weierud. Breaking German army ciphers. *Cryptologia*, 29(3):193–232, July 2005.

[223] A.A.R. Townsend. Genetic algorithms - a tutorial. 2003. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.3148&rep=rep1&type=pdf`.

[224] M.F. Uddin and A.M. Youssef. Cryptanalysis of Pointcheval's identification scheme using ant colony optimization. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC2007)*, pages 2942–2947. IEEE, September 2007. Digital Object Identifier 10.1109/CEC.2007.4424846.

[225] S. Vaudenay. Provable security for block ciphers by decorrelation. In D. Krob, C. Meinel, and M. Morvan, editors, *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 98)*, volume 1373 of *Lecture Notes in Computer Science*, pages 249–275. Springer, February 1998.

[226] S. Vaudenay. The decorrelation technique. September 2000. http://lasecwww.epfl.ch/memo/decorrelation.shtml. Accessed 18th June 2009.

[227] S. Vaudenay. Introduction to Decorrelation Theory On-Line Manual. July 2002. http://lasecwww.epfl.ch/memo/dec_manual.shtml. Accessed 18th June 2009.

[228] S. Vaudenay. Decorrelation: A theory for block cipher security. *Journal of Cryptology*, 16(4):249–286, September 2003.

[229] D. Wagner. The boomerang attack. In L.R. Knudsen, editor, *Proceedings of the Sixth International Workshop on Fast Software Encryption (FSE 1999)*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. IACR, Springer, March 1999.

[230] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, June 1994.

[231] Wikipedia. Data Encryption Standard — Wikipedia, the Free Encyclopedia. 2009. Online (http://en.wikipedia.org/w/index.php?title=Data_Encryption_Standard&oldid=298693027); accessed 26th June 2009.

[232] X-M. Zhang and Y. Zheng. GAC - the criterion for global avalanche characteristics of cryptographic functions. *Journal of Universal Computer Science*, 1(5):320–337, May 1995.