Environment-Independent Performance Analyses of Cryptographic Algorithms

Adam Dunn

The University of Western Australia,
Department of Computer Science & Software Engineering
35 Stirling Highway, Crawley WA 6009
Email: dunna02@csse.uwa.edu.au

Abstract

Generalising methods of measurement of performance for symmetric block ciphers is difficult. Cryptosystem performance is an issue that has become more important in recent years because of a trend towards more varied forms of communication requiring secure data transfer. Performance measurement suffers from a trade-off between accuracy and generalisation; accuracy in the experimental results is traded for the ability to extrapolate the data to other environments. Analysis of the atomic operations involved in ciphers can help to extrapolate environment-specific data to environments that have not yet been tested. The structure and reuse of similar operations in modern symmetric block ciphers aid in the construction of pictorial and language-based descriptions of the cipher algorithms. The goal of properly analysing the relationships between performance values of different environments can be reached through implementing these descriptions so that we can move freely between natural language, pictorial representations, and source code of a cipher. In this paper, I have detailed a solution to the problem relating to measurement, described my approach to implementing it, and evaluated this implementation with regards to functionality and usefulness.

Keywords: Advanced Encryption Standard (AES), Symmetric Block Ciphers.

1 Introduction

A *cryptosystem* is a system for encrypting a private message (*plaintext*) into a string (*ciphertext*), and decrypting to the original plaintext. The ciphertext retains all the information of the private message, but appears random to statistical testing. At the core of every cryptosystem is an algorithm that combines a *key* and the plaintext to achieve the ciphertext as an output. *Cryptanalysis* is the term given to the branch of cryptography concerned with mathematical analysis of the security of a cryptosystem. Performance analysis of cryptosystems has become more important to the cryptographic community recently, because of the increased need for secure communication in a much wider set of environments; especially in the emerging area of e-commerce, smart-cards, PDAs, and mobile and wireless communications.

1.1 Background

Most modern cryptosystems are built around *symmetric block ciphers*. Symmetric block ciphers are a specific form of cipher, where plaintext is encrypted a block at a time and is then decrypted using the same key. The difference between symmetric ciphers and asymmetric ciphers in terms of their operation is that asymmetric ciphers do not require both parties knowing the same secret key. Asymmetric encryption is also much slower than symmetric encryption. The predominant method of encryption is to use an asymmetric cipher to encrypt the key for a symmetric cipher that is then used in communicating a secret message.

The Data Encryption Standard (DES) was issued as a Federal Information Processing Standard in July 1977 (NIST 1977) and since 1999, DES is only used in legacy systems (NIST 1999*a*) because it has become insecure, due to developments in both hardware and mathematical cryptanalysis. DES belongs to a group of symmetric block ciphers known as Feistel networks. A *Feistel network* is traditionally built such that the text being encrypted is split into two halves. A function is applied to one half with the introduction of the key, and then the Boolean exclusive-or (XOR) operator is applied to the result of the function and the other half. The two halves are then swapped (Feistel 1973). Many modern ciphers are based on a generalisation of this structure, as proposed by Schneier in 1996 (Schneier & Kelsey 1996). Feistel structure is desirable in ciphers since encryption and decryption are structurally identical. In 1996, 44% of 1393 encryption products identified worldwide implemented the DES (Denning 1997). In early 1998, DES was broken in 56 hours (Landau 2000), confirming the obsolescence of DES. One ciphertext was decrypted by finding a 56-bit key using dedicated hardware.

In 1998, the National Institute of Standards and Technology (NIST) issued a challenge to the cryptographic community; proposing a competition to replace the defunct DES (NIST 1997). The Advanced Encryption Standard (AES) was the outcome of this competition. Fifteen algorithms were submitted to the first round of competition for the AES. The eventual winner of the competition was a cryptosystem developed by Vincent Rijmen and Joan Daemen. It was named Rijndael and the general structure of the cipher which forms the basis for the cryptosystem is a substitution linear transform network (SLTN). This is a form of a substitution permutation network (SPN) involving a layer of linear transforms in each round.

Interest shifted away from performance in 2002 because of the controversy surrounding the new information regarding possible attacks on two of the finalists of the AES competition (Courtois & Pieprzyk 2002, Ferguson 2001). There has always been a continual cycle of new cryptanalysis techniques being developed, causing designers of encryption algorithms to formulate more secure algorithms. A consequence of this cycle is that there is always a need for performance measurement of state-of-the-art encryption algorithms, because algorithms that push the boundary of security will inevitably require greater time and space complexities.

1.2 Motivation

Testing the performance of cryptographic algorithms is a restricted pursuit. The nature of these algorithms and the environments in which they are used is such that results in one environment cannot be easily extrapolated to another. When NIST requested public comment on round two finalists for the AES competition (NIST 1999*b*), a significant amount of testing was undertaken by the global cryptographic community within the following twelve months. The testing did not always provide convincing evidence on performance; various environments were used for testing of the first and second round finalists' algorithms, but the results of the testing did not always agree with each other. Although not all results are expected to agree, the differences between the conclusions of the papers formally submitted and accepted by NIST during the round two selection phase of the competition were substantial.

If the implementations of the different algorithms were standardised, we could expect a much higher correlation between experiments. However, standardisation cannot result in the ability to effectively extrapolate results from one environment to another; this is the problem I examined in this paper, by developing a method to analyse the attributes of encryption algorithms that are independent of environment. I hypothesise that this approach will give us the ability to more accurately predict the relative performance of encryption algorithms in many environments.

A solution to the problem associated with finding a method which measures environment independent attributes of encryption algorithms requires understanding what makes ciphers behave differently in different environments. Metrics which do not rely on environment should be considered as important complements to environment specific statistical measurements. In this way, we can use the raw performance results as a base for extrapolation and to substantiate the results of higher level assessments.

2 Literature Review

The most important feature of the AES is its ability to function in emerging technologies; standards are chosen to last for decades and as such, NIST was forced to anticipate the attributes that were important to extensibility of environment. NIST took an intelligent approach to this problem when public comment on the performance was requested in 1999 (NIST 1999*b*). This request spawned countless discussions and debate over five relatively similar encryption algorithms. A large amount of research into the performance of the five AES competition finalists was the result of the request and the ensuing debates. By comparing these papers, it is evident that many different approaches to performance testing can give a large set of results. However, there is no guarantee that the results will concur.

The final five algorithms of the AES competition were analysed by many cryptographers from around the world. MARS was an innovative design, using a wide range of operations and including rotations based on the data and the key (Burwick, Coppersmith, D'Avignon, Gennaro, Halevi, Jutla, Matyas, O'Connor, Peyravian, Safford & Zunic n.d.). RC6 is a simple design, also using data dependent rotations as a main security feature (Rivest, Robshaw, Sidney & Yin n.d.). Data dependent operations are detailed in later chapters of this paper. Rijndael has a straightforward design and tests show that it performs very well in hardware and software (Daemen & Rijmen n.d.). Serpent is designed conservatively in terms of security with a high number of rounds (32) (Anderson, Biham & Knudsen n.d.). Twofish is a versatile design, allowing for optimisations dependent on environment (Schneier, Kelsey, Whiting, Wagner & Hall 1998). Four of five of the finalists use a substitution box, known as an S-box. An S-box simply maps one number to another based on a table. A more formal definition of S-boxes appears in chapter three.

2.1 Performance of Encryption Algorithms

Bassham (Bassham 2000) and Schneier and Whiting (Schneier & Whiting 2000) present statistical analyses of the AES round two finalists. Bassham's affiliation is NIST and Schneier and Whiting are two of the inventors of the Twofish cryptosystem. Bassham's paper uses cycle-counting programs to measure performance on various processors, describing in detail the particular processor/hardware, operating system and compiler. Schneier and Whiting's paper attempts to provide an extensive statistical report on the five AES finalists based on key setup, raw encryption/decryption speeds, minimal secure variants (maximum insecure variants) and performance on eight-bit smart cards.

Despite the reference to measuring relative performance of algorithms instead of absolute performance, some results from the NIST report show big differences between the small ranges of machines on which the algorithms were tested. This means that it is unlikely that the results of the experiments could be extrapolated to environments of larger or smaller processing power or environments that are otherwise different.

In contrast to Bassham, Schneier and Whiting attempt to address a wide range of measures statistically, providing some performance details for the five algorithms on 32-bit and 64-bit CPUs coded in assembly. The results substantiate Bassham's conclusions in some ways:

- Serpent performs slower on all systems.
- Rijndael and Twofish perform relatively consistently across platforms.
- The two papers both agreed that Rijndael is one of a number of cryptosystems which could be used for widespread implementation.

The results of Schneier and Whiting's testing conflict with Bassham's results in the following ways:

- Twofish performs much better in the tests undertaken by the Twofish duo.
- Mars appears to perform considerably worse than Twofish and Rijndael in Schneier and Whiting's tests.
- The papers disagree on cryptosystems that performed well overall. Schneier and Whiting suggest that besides Rijndael, Twofish is the only other algorithm to perform well. Bassham suggests that RC6 performed well and that Twofish and Serpent were the two worst performers.

The conclusion that I draw from the conflicts seen in these two papers is that for these algorithms, exact performance values and environment specifics are inseparable; different hardware produces not only different results, but results that often directly contradict what is expected. The two testing environments described above are not especially far apart. We might expect large differences between Field Programmable Gate Array (FPGA) implementations and software implementations, but we would expect most results to match for testing environments using similar processors and operating systems.

2.2 Metrics Used in Measuring Performance

Evaluation metrics are discussed by Elbirt et al. in terms of their usefulness in testing an FPGA implementation of the AES (Elbirt, Yip, Chetwynd & Paar 2000). Some of the metrics considered by Elbirt et al. in their testing include throughput and hardware resource cost metrics such as logic gate area measurement and configurable logic block (CLB) slice measurement. Elbirt et al. introduce a new metric called throughput per slice (TPS); defined as the rate of encryption divided by the number of CLB slices.

Although Elbirt et al. only use one FPGA (the Xilinx Virtex XCV1000BG560-4), they discuss the types of operations performed in the five finalist algorithms for the AES competition and the expected difficulties in implementing some of these operations in hardware. One operation that is generally expected to require the most time and hardware resources is the modulo 2^{32} operation. Table 2 shows that MARS, RC6 and Twofish use this operation in their respective algorithms. One hypothesis that can be automatically made about the five algorithms in terms of their FPGA implementations is that the two algorithms that do not use addition modulo 2^{32} will perform better. This hypothesis is supported by their experiments and they concluded that Rijndael and Serpent indeed showed the best performance. This is clear evidence that an in-depth analysis of the operations used in the algorithms can provide us with information useful in determining relative performance in environments that have not had implementations of the algorithms built for them.

The NSA (Weeks et al.) published a 'Final Report' on its findings of the performance of the five round two finalists of the AES competition in hardware. Their target applications included iterative and pipelined architectures. Weeks et al. decided to publish only the results for each experiment but not make any judgements about overall performance. The metrics that are used in these experiments include area, throughput, transistor count, I/O required, key setup time, algorithm setup time and time to encrypt/decrypt one block. These metrics range between the extremes of environment dependence. A metric such as time to encrypt one block is highly sensitive to the environment in which the algorithm is implemented, whereas metrics like area and transistor count are independent of how the algorithm is implemented. The associated problem with the area and transistor count metrics is that they only relate to hardware implementations. Comparisons between performance of encryption algorithms would be more effective if there was a method of testing the algorithms independent of their environment and such that the results are applicable to a wide range of applications.

3 Generalising Modern Cryptosystem Algorithms

To be able to comment effectively on the relative performance of encryption algorithms based on the atomic operations, it is important to standardise the set of operations that are being used in the algorithms being tested. This is not a laborious task since the algorithms are similar in nature, the operations are mathematically defined, and many are common to a large majority of modern symmetric block ciphers.

The types of algorithms that are of current, practical use in cryptosystems generally fit into two categories of ciphers; specifically, they are either a form of Feistel network or a network of substitutions, and permutations or linear transforms. There are other encryption algorithms based on other concepts that have been proposed by researchers from within the community, but the most are either theoretical, impractical, or both.

NIST received 21 submission packages for the AES competition, of which 15 satisfied the requirements of the competition. These 15 algorithms are a good guide to where symmetric ciphers stood at the time of the competition (1997-1999); they represented the best algorithms that were of practical use in common commercial and private applications. All but one of the 15 algorithms accepted into the first round can be classified as Feistel networks, substitution-permutation networks (SPNs), or substitution-linear-transform networks (SLTNs). The observation made about classifying encryption algorithms is a useful one, because it allows for the definition of a set of functions that can completely represent the operations that comprise an algorithm.

The number of functions required to completely represent the operations that comprise a majority of modern ciphers is small enough to be feasibly implemented, but several of these operations cause some difficulties in terms of generalisation. The difficulties include variance in ways of implementing the same operation, dependence on outside data, and operations that require large lookup tables or a set of lookup tables.

3.1 S-boxes

An S-box is defined as a non-linear mapping $\{0,1\}^x \rightarrow \{0,1\}^y$ where x is the number of input bits, y is the number of output bits, and $x > y$ (Hendessi, Gulliver & Shieke 1997). An S-box is usually implemented as an m by n matrix where $n = 2^y$ and $m = 2^{x-y}$. An S-box represented as a matrix in this fashion is called an m by n S-box.

Research into S-boxes can be categorised into the two main areas of analysis and design. Encryption algorithms (especially SPNs and Feistel networks) are generally designed around the principles of confusion and diffusion. *Confusion* is the result of substitution and *diffusion* is the result of permutation. S-boxes are the usual way to provide confusion in modern ciphers. The terminology regarding substitution-permutation ciphers was formulated during the 1970s (Kam & Davida 1979) and since this time, S-boxes have been subject to much research in both analysis and design.

Nonlinearity (Xu & Heys 1997), randomness, and avalanche (Feistel 1973) are some of the design criteria of S-boxes. In 1985, Webster introduced the notion of *Strict Avalanche Criterion* (SAC) (Webster & Tavares 1985). The SAC is that any changes in the ciphertext should affect all the bits in the plaintext (when decrypting) in a random fashion. If it satisfies this criterion, it is said to be *two-way complete.*

The differences between the design of S-boxes shown in Table 1 are interesting because all algorithms are designed to provide a fair trade-off between performance and security. In general, a smaller S-box is more efficient and a larger S-box is more secure. Different algorithms specify different implementations for S-boxes, and it is clear that some S-boxes are stronger than others. Since there are differences in security margins between different S-boxes, there is feasibly an S-box (or a set of S-boxes) that have optimal security. One hypothesis regarding S-box design might be that a single (optimal) S-box could provide sufficient security for all algorithms. However, this hypothesis is not substantiated by current trends in S-box design research. The major factor in the security margin of S-boxes is time; security flaws in practical implementations of encryption algorithms that have been broken through analysis of S-boxes have required time to be discovered. For example, features

Table 1: S-box Design in AES Finalists

Cipher	S-Box Size	Method of Development
MARS	512 32-bit words	Generated using a psuedorandom and modified constants until specific criteria were achieved (Burwick et al. n.d.).
Rijndael	256 8-bit words	S-boxes defined by operations in GF(2^8), a matrix multiplication, and an XOR operation (Daemen & Rijmen n.d.).
Twofish	4 8-by-8 bit S-boxes	Key-dependent - built from 8-by-8 bit permutations and key material (Schneier et al. 1998).
Serpent	8 S-boxes with 32 16-bit words	Uses DES S-boxes and substitutes until desired properties are achieved for each of the 8 (Anderson et al. n.d.).

of the DES S-boxes were exploited in the differential cryptanalysis of the algorithm in 1990 (Landau 2000).

S-boxes are believed to provide the majority of security in encryption algorithms. Recently however, properties of specific S-boxes have been used to theoretically break several algorithms. The S-box construction of Rijndael and Serpent have been represented algebraically (Ferguson 2001) and this result has led to claims that both algorithms can be broken in close to polynomial time (Schneier 2002, Courtois & Pieprzyk 2002).

3.2 Galois Field Operations

A Galois field is a field of finite cardinality (order). A Galois field is made up of the integers $[0,\ldots,q-1]$ where q is a power of a prime number. A Galois field is closed under modulo p addition and multiplication. The representation for Galois fields that allows for modulo multiplication and modulo addition is as a polynomial. This representation is generally reduced to a binary vector of a specific format when used in ciphers.

Operations in Galois fields can be implemented using bit operations in modern ciphers, but they require additional knowledge in some cases. Galois fields are used more commonly in ciphers developed in the last few years; Rijndael being the best known of these.

Modulo multiplication under GF(2^m) can be computationally expensive when implemented in either software or hardware. The first operation is to multiply each of the terms in the second factor by the first factor and take the sequential XOR of the set of results. The next step is to divide the product by the primitive polynomial. The *primitive polynomial* is the irreducible polynomial of degree equal to m. A polynomial is *irreducible* if it has no other divisors besides 1 and itself. The irreducible polynomial for GF(2^8) is $(X^8+X^4+X^3+X+1)$. As a binary vector, this is [100011011]. The result is a polynomial of degree less than m (in this case, less than 8).

3.3 Data Dependent Operations

Data dependent operations are not used in the majority of the encryption algorithms discussed in this paper, but they are an interesting feature in terms of security. RC6 and the Hasty Pudding Cipher are particular examples of modern ciphers which include data dependent operations in their encryption algorithms.

4 A Solution to the Accuracy versus Generality Problem

Solving the accuracy versus generality problem requires an intelligent mix of testing; comprising both environment specific experiments and assessment of the ciphers using metrics which are not affected by environment. The results should include values for raw performance, as well as a full representation of the atomic operations of the cipher (which are generally not affected by environment). This allows an effective platform from which to extrapolate data into unknown environments.

Symmetric Block Ciphers are amenable to a directed graph structure because of the reuse of functions and the way data flows through the set of functions. Cryptosystems and the ciphers that include them are represented in many ways: using natural language, as mathematical equations, or pictorially. Perhaps the most intuitive of these pictorial representations is the directed graph. Figure 1 shows a single round of DES (of which there are 19 in the algorithm), demonstrating visually how a Feistel network works on a local level. Figure 2 shows the overall structure of DES. An important process in accurately describing ciphers is to create a standard for the pictorial representation, ensuring that all details of the cipher are represented in the picture.

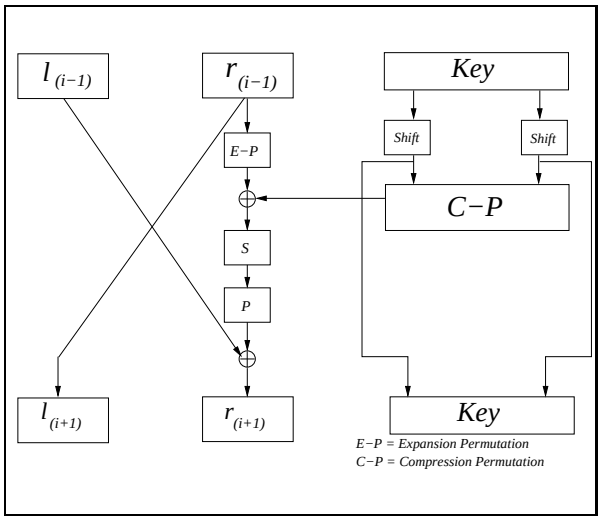

Figure 1: Pictorial Representation of a single DES round

4.1 Structure Of The Solution

The steps that I took in designing a solution for standardising the representations of ciphers included defining the structures that hold all the information that comprises the encryption algorithm. I extend the general definition of a directed graph to contain all the essential information concerning a specific cipher. A directed graph is defined by a set of nodes (boxes) and edges (channels) with the order that the edges are defined implying the direction of the edge (Sedgewick

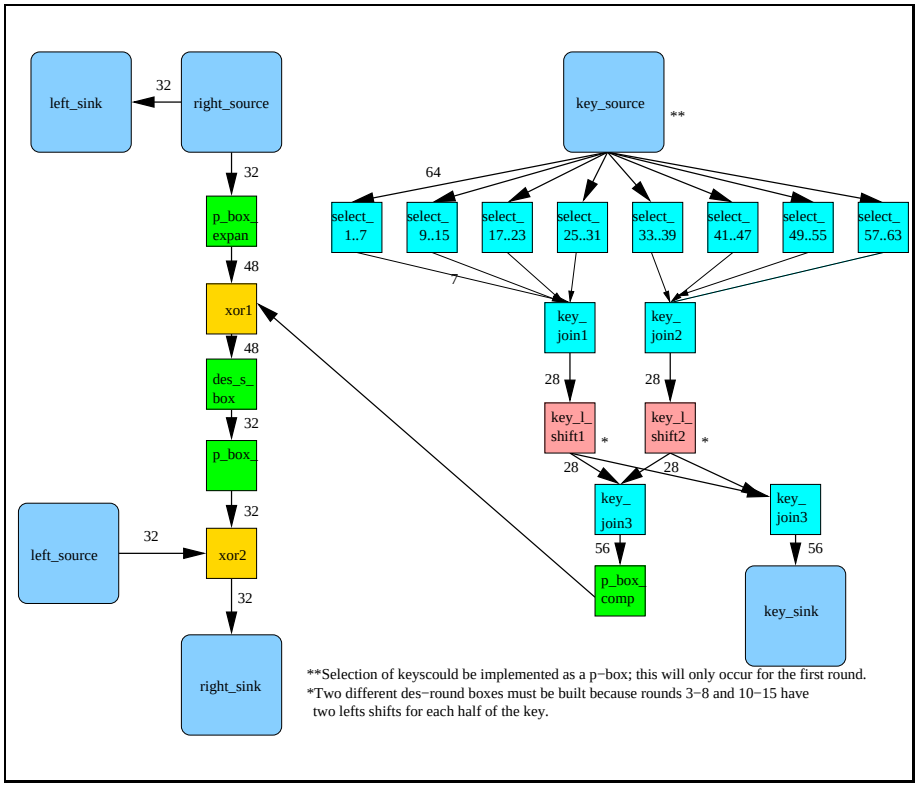

Figure 3: Extended DAG structure representation of a DES round

2002). Boxes and channels can be defined more specifically to include more information about the operations involved inside a box and the information that they are capable of accepting.

A channel is defined to have the following properties:

- one origin box,
- one goal box,
- an index (included to hold some extra information about its content), and
- a width (integer) specifying the number of bits which can pass through it.

The channel appears as a directed line from the originating box to the goal box, with the thickness of the arrow directly related to the number of bits that can pass through it. The index used to specify some extra information about how the data flowing through the channel should be used in the goal box appears as a string next to the channel.

A box is defined to have the following properties:

- a type (string) specifying the operation which the box carries out,
- a set of inputs (integer indices to channels), and
- a set of outputs (integer indices to channels).

The box's type also provides more information about the box's atomicity and the appearance of the box in terms of its colour and label. The set of inputs appear as the channels connected into the box and the set of outputs appear as the channels emanating from the box.

A survey of a large proportion of the modern block ciphers that require analysis results in a set of functions or operations which are included in ciphers. These operations (listed in table 2) can be mapped directly to the extended structure proposed in this section as box types. The problem associated with selecting the set of operations to be mapped into the extended structure is twofold. Firstly, the set of operations must be finite and therefore not all operations can be specifically incorporated; secondly, the set of functions must be minimised for efficiency in implementation. Table 2 shows that the number of operations used most frequently is small and there are few anomalous operations.

Recursion plays an important role in the extended structure proposed in this section. The structure of the ciphers described in the previous sections is highly modular. Each round of any cipher is repeated a number of times (usually between 6 and 32) and can be removed as an individual structure, essentially allowing a hierarchical structure. Individual parts of ciphers are also repeated within rounds, within other parts of the same cipher, and in other ciphers.

The pictorial representation resulting from the set of definitions describes a cipher completely (using references to S-box tables only). An example shows how a typical pictorial representation and the standard definition of a directed graph is modified to include the extra information (Figures 3 and 4).

4.2 Restrictions and Difficulties in How Problems are Modelled

A topological sort can be applied to any directed acyclic graph (DAG) to relabel or rearrange the boxes so that all the directed edges point in a forward direction. A topological sort is especially useful in determining the order in which boxes need to be processed so that each box is processed before all the boxes to which it points (Sedgewick 2002). To apply a topological sort to the defined structure, a restriction must be applied to the graph, it must be acyclic. In terms of how a cipher is structured, this has one major implication. Each round must be stored sequentially. Since the definition of the structure already includes recursion, this problem is minimised.

Table 2: AES Round Two Operations Used

Algorithm	XOR	Add/Sub (2^{32})	Fixed/Var. Rotation	Multip . (2^{32})	Galois Multip.	S-Box
MARS	Yes	Yes	Yes	Yes	No	Yes
RC6	Yes	Yes	Yes	Yes	No	No
Rijndael	Yes	No	Yes	No	Yes	Yes
Serpent	Yes	No	Yes	No	No	Yes
Twofish	Yes	Yes	Yes	No	Yes	Yes

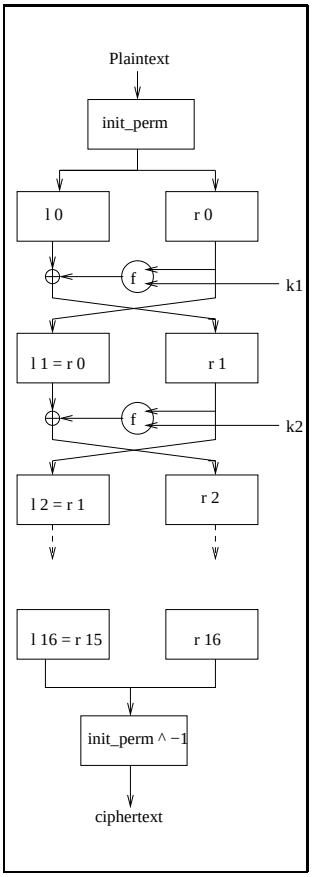

Figure 2: Pictorial Representation of DES

The process of replacing a looped set of operations (especially sets of rounds) by a sequential set of equivalent operations in a cipher is known as loop unrolling (Nechvatal, Barker, Bassham, Burr, Dworkin, Foti & Roback 2001).

The perceived difficulty in generalising S-boxes causes some restrictions on the algorithms that can be modelled. Building a system including a specific S-box is easy to do, provided that the user has knowledge of the lookup table that constitutes the S-box implementation. Using data-dependent S-box generation (or not having knowledge of the S-box being used), is not easily constructed using the structure defined above. Fortunately, a majority of the algorithms discussed in this paper only used predefined S-boxes. Twofish uses S-boxes that are key-dependent; a feature which the designers say enhance security (Schneier et al. 1998).

The specific inclusion of data dependence would require dual implementations of the same functions. Twofish uses data-dependency to generate S-boxes (Schneier et al. 1998). Schneier et al. specify S-boxes using a function (h) which takes two inputs: a 32-bit word X and a list L (of length k) of 32-bit words. The function acts on X using data from L, based on its length (k). To implement this particular set of operations, we would need to (at least) extend the notion of channels to include switches based on *if* statements. For example, we may want to send the output of a box as the input of an XOR box if the cardinality of the input set is above a threshold or bypass the XOR box if the cardinality is below the threshold (*cardinality* is number of 'ones' in the binary representation of the set of bits). The complexity added to the DAG representation and the associated topological sort makes this extension impractical in the implementations current form.

4.3 Metrics That Can be Included in the Solution

The most important part of the implementation of this solution is the inclusion of useful metrics. Most papers dealing with evaluation of performance deal with metrics such as throughput and latency; many of them also deal with more specific metrics including key-setup time, transistor counts, and areas (Weeks, Bean, Rozylowicz & Ficke 2000, Nechvatal et al. 2001, Schneier & Whiting 2000). I have not seen an extensive evaluation of the atomic operations in the ciphers. The structure in which the algorithms have been described allows for a large number of metrics, especially those that do not depend on the environment.

The number of invocations of each operation is an important factor in determining relative performance between environments. A series of submissions were made to NIST after they requested public comment on the round two finalists of the AES competition. Sybrandy communicated to NIST a statement about RC6 and Twofish, pointing out that RC6 relied heavily on multiplication whereas Twofish used two tables (Sybrandy 1999). In this case, we know that RC6 would not be a good choice in an environment where multiplication was expensive (for example, smart cards), and Twofish would not be good in an environment in which memory was of critical importance.

I propose that a table listing the frequency of operations per block of plaintext encrypted or decrypted, would not only allow decisions to be made about different environments, but that it would also aid in the extrapolation of time-based data into different environments. One way to extract useful information more quickly would be to maintain a set of scores for each atomic operation in different environments. The *scores* may be affected by average expected times taken for the particular environment, the financial cost associated with implementing the operation in hardware (for hardware environments), and the expected load on memory for the specific environment (or any combination of these three measures).

5 Method of Development

Implementing the extended graph structure and topological sort to the point at which ciphers are operable, requires ease of movement between the forms of

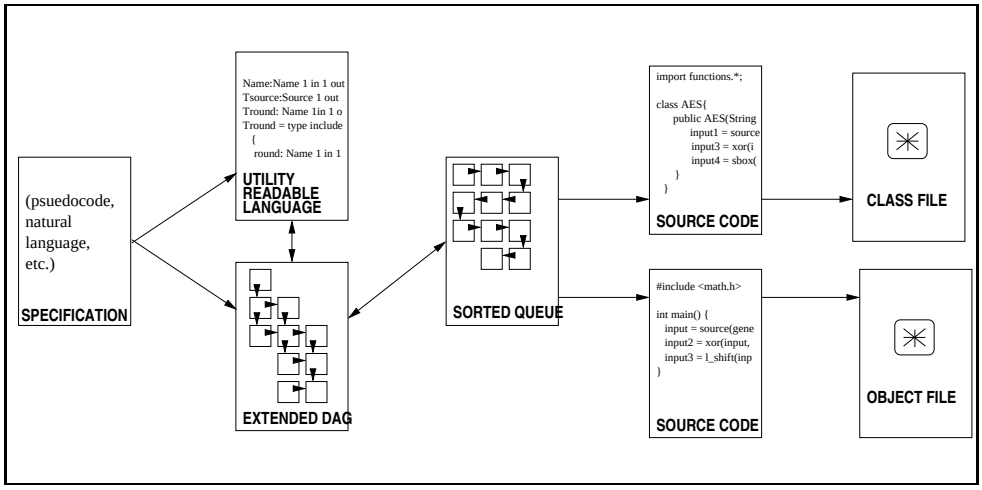

Figure 5: Data Transition from Specification to Class File

representations of the ciphers. The flow between designers' specifications to reported results is detailed in this chapter and illustrated in Figure 5. The task of implementing this 'free flow' between representations requires the design and implementation of a parser for the standard, a library of 'boxes', the data structures to hold them, a source code writer, and a graphical user interface (GUI). Within my implementation there are 8 packages, 50 Java class files, a library of standardised ciphers, and corresponding libraries of tables and runnable cipher classes.

Testing the functionality of the implementation is different to testing the usefulness of the solution to the problem of measuring encryption algorithms independent of environment. A preliminary implementation can predict some of the abilities and difficulties that a full implementation may have. Finding these abilities and difficulties can be done by analysing the boundaries of what ciphers can be implemented within the system, and trying to replicate expected results for simplified problems.

5.1 From Designers' Specifications to Standard Specification

Designers' pictorial representations of ciphers can rarely be read as stand-alone specifications. The usual reason for including a pictorial representation of an algorithm is to give the reader of the specification an overview of the algorithm at an abstracted level. Simple operations such as XOR, shifts, and rotations can easily be shown as a symbol or a labeled box, but more complex operations such as substitutions and permutations need to be represented at a lower level as a number of operations or detailed elsewhere.

By restricting the set of functions available and defining their inputs, a standard specification will always be completely covered by the set of implemented functions. Each operation and the corresponding set of arguments defines exactly the behaviour of the every box. Building an optimal set of operations is intuitive; too many operations or inputs may result in a less readable language, and not enough operations and the number of ciphers that can be represented becomes too small. The set of functions consist of modulo addition, join, left rotation, left shift, Galois multiplication, modulo multiplication, name, right rotation, right shift, s-box, select, sink, source, and XOR.

5.2 Between Standard Specifications and Structured Data

Storage of the extended acyclic graph employs the standard specification in the form of a human readable/writable language in which all boxes and channels are stored with their attributes and arguments. A more compact and efficient storage method is possible, but the benefit of a human readable specification outweighs the need for compact storage. For an initial implementation, a small human writable language is a good point for human intervention (the point at which we can input new cipher specifications). A 'drag-and-drop' method of implementing new ciphers would be desirable, but the implementation of this requires too much time.

An *adjacency matrix* is a 2-dimensional array of Boolean values where a row-column value is set to true if the box associated with the index of the row is connected to the box associated with the index of the column. The adjacency matrix forms the nucleus of the data structure, specifying all the connections within one structure. Each *structure* represents a single layer of a particular cipher; a set of boxes connected only by channels (as opposed to the link associated with recursive boxes and its children) and bounded by sinks and sources. Each structure has its own adjacency matrix, and a collection of external links (to other structures) defined by named boxes.

A *topological sort* is a sorting method that is applied to a directed acyclic graph such that each node is placed in a list where all of its children are always listed after it (Sedgewick 2002). Topological sort is written as a recursive function in this implementation. It allows us to specify an order of execution for each structure. I extend this method to specify an order not only for each structure individually, but also interlacing all structures within the cipher which is operating.

5.3 From Structured Data to Compilable Source Code

Generating a set of sequential instructions (from a set of boxes and channels) that a compiler will recognise can be implemented using a simple mapping taking the box type and number of inputs and simply outputting a function call per line. For each box, a *ClassWriter* object is passed the function name associated with the box type, the index of the box, a set of input indices, and possibly some other arguments (such as the primitive polynomial in the case of a Galois multiplication box).

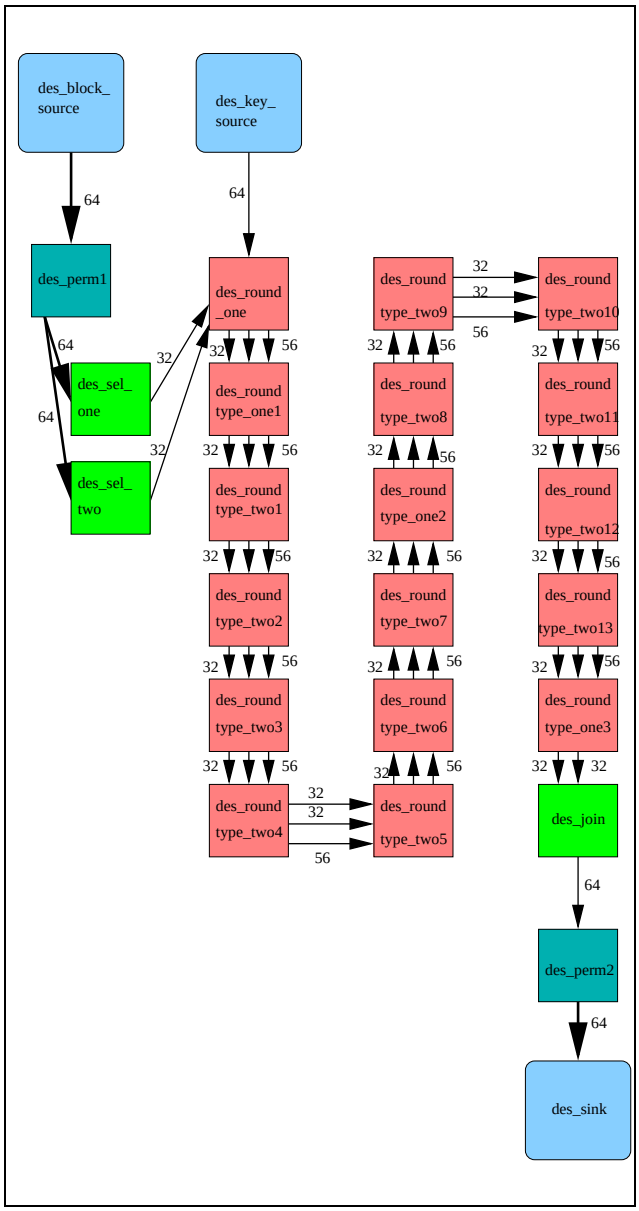

Figure 4: Extended DAG structure representation of DES

Each line is added to the particular function, with specific cases for 'name' boxes (which generates a new function), source boxes (which initialises the input for the function), and sink boxes (which end the function with a return statement). The headers (import statements and constructor) for the file are general, using the global name of the standard specification for naming.

Testing the ability of my implementation to completely represent the set of five algorithms described in this paper is the first important step in validating the solution. Validating the usefulness of the solution involves analysing the results pertaining to the atomic operation counts and matching these results to a large set of raw performance results for different environments.

6 Evaluating the Implementation

The effectiveness of my implementation of the solution to completely represent the set of five algorithms discussed in this paper are varied. The usefulness of using the atomic operations included in my solution to predict performance in specific environments is not substantiated by my implementation of the solution (using toy ciphers with specific known properties).

6.1 Representing Ciphers Completely

Only three of the five algorithms were implementable because I did not implement data-dependent operations as a standard function because of the increase in complexity of the solution. There were difficulties in implementing the algorithms in concise efficient ways in several cases due to the atomicity of the functions; complex sets of boxes and channels were needed in place of lookup tables due to the restrictions I placed on the the set of operations included in this implementation.

S-boxes caused different problems based on their design. Designs for S-boxes that can be implemented dynamically (such as in Rijndael or Twofish) usually required operations that were not included in the standard set of functions or required external data anyway (this would cause redundant functions because the final S-boxes could be stored instead). S-boxes for which the details of the development were not published or could not be implemented dynamically required external data stored in the form of tables.

Rijndael is fully implementable within the current implementation. Two difficult operations (S-box and GF(256) multiplication) are implemented by passing arguments (such as files containing tables or binary representations of primitive polynomials) to the language parser via the standard specification.

The implementation of the Rijndael S-box demonstrates a trade-off between generality of operations and efficiency of operations. The Rijndael S-box is a single array of 256 words, 8 bits in length. It can be implemented as a lookup table where the filename of where the values are stored is passed to the box as an argument. A much more convenient way of implementing the Rijndael S-box would be to simply apply the GF(256) operations, the matrix multiplication, and the XOR operation. This method would require the inclusion of multiplicative inverse (within a Galois field) operation in the library of standard functions. The general solution trades generality for performance and reliability in this case.

Since the Rijndael S-box is simply a list of 256 8-bit words, there is no need for extra operations to identify the proper indexing. The current implementation of the S-box function takes the number of inputs (in this case it is one), and assumes that the array contained in the file specified is of that dimension. It then uses this to find the correct value.

The Galois field operation is achieved successfully by passing the primitive polynomial for the width of the channel leading into the box containing the operation. The standardised representation which facilitates the building of the box for this operation is as follows: The arguments that are passed to the box relate to the size of the Galois field (in the case of Rijndael it is $2^8 = 256$), and the primitive polynomial associated with this size (which is $X^8+X^4+X^3+X+1$ for this size). There is some redundancy in this input, since the primitive polynomial will always be the same for each value representing the size.

Serpent is fully implementable, albeit very complex to write in this implementation of the general solution. The initial and final permutations can only be implemented as a complex graph of selects and joins. This is certainly not easy to implement, but it does represent the permutations correctly and can provide more details about the atomic operations than if the permutation operation was implemented as a look up table in the same way as an S-box.

The permutation operation that exists in many modern encryption algorithms is better implemented as a set of atomic operations because it gives more information about the operation than could be possible if the operation was defined as atomic in the solution we have designed. This is particularly necessary when comparing two different algorithms; there may be significant differences between overall results due to differences in how join and select operations (which are generally the atomic operations that make up a permutation operation) are performed in different environments.

Twofish is a special case because all of the functions it requires are available to it in my implementation of the general solution, but the method of construction for the S-box cannot be implemented using the solution functions. There are no feasible sets of operations that could allow for the checking of each generated S-box for its security characteristics. Twofish can be implemented in terms of reporting on the atomic operations, but any implementation within the general solution would not be an effective implementation of Twofish because fixed S-boxes (lookup tables stored in external files) would have to be used.

RC6 was not implementable due to the need for data-dependent operations. Since RC6 does not use Galois operations or S-boxes, the algorithm is easily represented by the general solution if the data dependency rotations are changed to fixed rotations.

MARS is often described as being interesting because it uses such a wide range of functions. MARS was designed to utilise all the strongest tools that were currently available for designing ciphers. For this reason, MARS suffers from the same problems in terms of generalisation as the previous four algorithms, including a single S-box and data dependent rotations. Multiplication is generally not included in encryption algorithms because it is historically, prohibitively expensive. Burwick et al. suggest that multiplication is no longer an expensive operation because most architectures support a quick multiply instruction (Burwick et al. n.d.). If the data-dependent rotations are modified to be fixed, then MARS is the only algorithm to take advantage of the multiply function. This may be because the algorithms entered into the AES competition were designed to perform consistently well in many environments. Algorithms that are only designed to perform well in specific environments might use the multiplication operation if the particular operating environment provides cheap multiply instructions.

6.2 Atomic Operation Counts as Metrics

Counting atomic operations appears to be a useful pursuit when analysing performance of encryption algorithms. The method for testing this hypothesis is to test a particular algorithm for which the expected results are quite clear. I chose to implement a simplified version of the Serpent algorithm because the testing involving comparison between software and FPGA implementations of this algorithm all agree that the algorithm performs relatively better on the FPGA (as opposed to several of the other algorithms described above in the same pair of environments).

The Serpent algorithm uses an initial and final permutation simply as a method to simplify an optimised implementation of the cipher and improve computational efficiency. After removing these permutations, the structure is 32 rounds of a key-mixing operation, a pass through S-boxes, and a linear transformation (Anderson et al. n.d.). The 32 rounds use 8 different S-boxes.

If the Serpent algorithm is simplified to use the same key over all rounds and use only one S-box, the simplified version retains all the atomic operations of Serpent (excluding the subkey generation operations) and it is easier to implement and understand. The permutations are understood to be a constant number of select and join operations. By comparing the specific results of software and FPGA environments using my implementation of the solution shows that Serpent would perform better in a software environment compared to an FPGA implementation. The results indicate that this is caused by the relative expense of the S-box operation in the FPGA environment compared to the software environment. In interpreting these scores, some issues need to be addressed. Every score must be comparatively accurate for the environment that it is in, as well as comparatively accurate to the scores for the same operation in all other listed environments. An accurate table of values could be attained from sourcing data regarding raw performance of atomic operations. One difficulty is in knowing how specific S-boxes will perform in other environments. To get an accurate score for an S-box we need to know how many inputs (how many indexes it requires) it has and the actual size of the S-box lookup table.

Another source of problems for this method of analysing performance is the lack of suitable metrics. Including other metrics is important to the success of this particular implementation of the solution. Specifically, including a measure of parallelism in the cipher is important because the implementation is sequential and environments using bit-slicing or hardware based implementations are affected greatly by the ability to execute the operations in parallel.

In my opinion, these results are preliminary at best. More testing and more information regarding atomic operations in specific environments needs to be acquired before this implementation can provide effective analysis of environments for which we do not have implementations. The specific use of this implementation of the solution is much more suited to comparing different algorithms (or modifications of the same algorithm) as opposed to comparing two environments for the same algorithm.

7 Conclusions and Further Work

Generalising encryption algorithms and designing a utility to take advantage of these general statements to produce some useful results is mainly an intuitive exercise based on a wide range of computer science and mathematical topics. The design and evaluation of cryptosystems has become a popular area of research in recent times mainly due to the global cryptographic community's involvement in the AES competition. Leading cryptographers and outspoken cryptosystem inventors generate a large amount of discussion on both the design of new algorithms as well as the cryptanalysis of current ciphers. The commercial implementation of cryptosystems in a wider range of environments has also contributed to the upsurge of popularity in the field of cryptography.

Formal descriptions of encryption algorithms have been achieved in many forms. These descriptions range from natural language descriptions to physical maps of FPGAs. Perhaps an area of neglect in this field of computer science research has been how effective these descriptions are. It is easily evident that we can discern differences between these descriptions in terms of their expressiveness. A highly statistical measure of efficiency of a specific implementation of an algorithm on one operating system using one set of hardware can provide good results using many metrics, but the results that are generated apply only

to that particular environment and cannot be accurately extrapolated to different environments. A natural language description of an encryption algorithm can describe its operation in a wide set of environments, but cannot give accurate information on performance.

7.1 Conclusions About the Evaluation

I have chosen to approach this problem by generalising a representative group of known ciphers and implementing the solution that could describe the associated algorithms in an intuitive and visual-based structure. The implementation of the solution I developed can be used to provide a link between the two extremes of performance measurement (accuracy and the ability to extrapolate) so that we can say more about the functionality of cryptosystems in different environments. The utility provides this link by assessing the properties of the algorithms which apply to all environments; the operations which are atomic (in terms of how algorithms can be implemented in a large set of environments).

However, the implementation is limited in several respects. A measure of parallelism is an important inclusion to the implementation, because there is currently no method of measuring the relative behaviour of parallel architecture implementations. The inability to represent data-dependent operations is also a limitation, because a significant proportion of the current ciphers include these kinds of operations in their specification.

References

Anderson, R., Biham, E. & Knudsen, L. (n.d.), 'Serpent: A proposal for the advanced encryption standard'.

Bassham, L. E. (2000), Efficiency testing of ANSI C implementations of round 2 candidate algorithms for the advanced encryption standard, Technical report, Computer Security Division Information Technology Labratory National Institute of Standards and Technology.

Burwick, C., Coppersmith, D., D'Avignon, E., Gennaro, R., Halevi, S., Jutla, C., Matyas, S., O'Connor, L., Peyravian, M., Safford, D. & Zunic, N. (n.d.), 'Mars — a candidate cipher for aes'.

Courtois, N. T. & Pieprzyk, J. (2002), 'Cryptanalysis of block ciphers with overdefined systems of equations'.

Daemen, J. & Rijmen, V. (n.d.), 'Aes proposal: Rijndael'.

Denning, D. E. (1997), 'Encryption policy and market trends', http://www.cs.georgetown.edu/ denning/crypto/Trends.html.

Elbirt, A. J., Yip, W., Chetwynd, B. & Paar, C. (2000), An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists, *in* 'Third Advanced Encryption Standard (AES) Candidate Conference'.

Feistel, H. (1973), 'Cryptography and computer privacy', *Scientific American* **228**(5), 15–23.

Ferguson, N. (2001), 'Unpublished', http://www.macfergus.com/niels/dmca/cia.html.

Hendessi, F., Gulliver, T. A. & Shieke, A. U. (1997), Large s-box design using a converging method, *in* 'IEEE International Symposium on Information Theory', p. 177.

Kam, J. & Davida, G. (1979), Structured design of substitution-permutation encryption networks, *in* 'IEEE Transactions on Computers', Vol. C28, pp. 747–753.

Landau, S. (2000), 'Standing the test of time: The data encryption standard', *Notices of the American Mathematical Society* pp. 341–349.

Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J. & Roback, E. (2001), 'Report on the development of the advanced encryption standard (AES)', *Journal of Research of the National Institute of Standards and Technology* **106**(3).

NIST (1977), 'Data encryption standard', FIPS 46-1.

NIST (1997), 'Announcing request for candidate algorithm nominations for the advanced encryption standard (AES)', Federal Register.

NIST (1999*a*), 'Data encryption standard', FIPS 46-3.

NIST (1999*b*), 'Request for comments on the finalist (round 2) candidate algorithms for the advanced encryption standard (aes)', Federal Register.

Rivest, R. L., Robshaw, M. J. B., Sidney, R. & Yin, Y. L. (n.d.), The RC6 block cipher.

Schneier, B. (2002), 'Crypto-gram newsletter', http://www.counterpane.com/crypto-gram.html.

Schneier, B. & Kelsey, J. (1996), Unbalanced feistel networks and block cipher design, *in* 'Fast Software Encryption, Third International Workshop Proceedings', pp. 121–144.

Schneier, B., Kelsey, J., Whiting, D., Wagner, D. & Hall, C. (1998), On the twofish key schedule, *in* 'Selected Areas in Cryptography', pp. 27–42.

Schneier, B. & Whiting, D. (2000), A performance comparison of the five aes finalists, *in* 'Third AES Candidate Conference'.

Sedgewick, R. (2002), *Algorithms in C, Part 5, Graph Algorithms*, 3 edn, Addison Wesley.

Sybrandy, C. (1999), 'In response to a notice in the september 15, 1999 Federal Register (Volume 64, Number 178; pages 50058-50061)', http://csrc.nist.gov/encryption/aes/round2/pubcmnts.htm via email.

Webster, A. & Tavares, S. (1985), On the design of s-boxes, *in* 'Advances in Cryptology – CRYPTO '85', pp. 523–534.

Weeks, B., Bean, M., Rozylowicz, T. & Ficke, C. (2000), Hardware performance simulations of round 2 advanced encryption standard algorithms, *in* 'Third Advanced Encryption Standard Candidate Conference', pp. 106–120.

Xu, J. & Heys, H. (1997), A new criterion for the design of 8x8 s-boxes in private key ciphers, *in* 'IEEE Canadian Conference on Electrical and Computer Engineering (CCECE '7')'.