A Cost/Performance Study of Modern FPGAs in Cryptanalysis

Ian Howson (ian@ianhowson.com)

Supervisors: Matt Barrie and Craig Jin

School of Electrical and Information Engineering
University of Sydney

Bachelor of Engineering (Software Engineering)

October 2003

Abstract

Exhaustive key search attacks will always be successful against any symmetric cipher given enough time. Many past and present deployments of cryptography are not strong enough to withstand a key search attack against a moderately funded adversary. This thesis compares the cost and performance of FPGA and software-based approaches to key search. This information is useful when assessing the security of cryptographic deployments and ciphers.

Implementations of cipher-independent FPGA key search machines yield performance estimates for DES and RC5. Price/performance comparisons within CPU and FPGA families help to determine what the cheapest, fastest devices are. These results show that past performance estimates were too high and that the EFF DES cracker will cost very little to build using modern FPGAs.

The thesis also describes a framework that estimates FPGA resource consumption for any symmetric cipher. This makes use of data describing the resource consumption of each primitive cipher operation. Ciphers can be classified to determine if they will perform well using hardware or software approaches.

Discussion on regulatory issues within Australia and the United States place the results into perspective and show that restrictions effectively destroy the intent of cryptographic protection.

Statement of Achievement

- Conducted a literature review of previous hardware and software key search efforts
- Generalised the design of any key search machine
- Designed, implemented and tested two generic FPGA key search machines
- Implemented and tested FPGA modules for the DES and RC5 symmetric ciphers
- Implemented and tested an FPGA statistical comparator that can be "trained" to recognise text types
- Conducted a number of DES and RC5 key search benchmarks for common CPUs
- Analysed and formalised the factors that affect exhaustive key search
- Produced a framework which can be used to estimate the FPGA resource usage for a cipher based on its algorithmic description
- Compared Xilinx FPGA devices, AMD and Intel CPUs to determine where the optimal price/performance points lie
- Compared my performance results with those in other works and determined the cost to replicate some past efforts with modern devices
- Examined regulatory issues relating to encryption and high-performance computing
- Suggested directions for future work

	.....	.....
Signed	Ian Howson	Matt Barrie

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The use of cryptography is growing rapidly with the adoption of computer technology. The design of cryptographic ciphers is still not well understood; we cannot prove the security of an algorithm. Currently, the only way to be sure of the security of an algorithm is to study it for a long period of time and use the absence of attacks as evidence confirming its security.

All ciphers are vulnerable to an *exhaustive key search* attack. An attacker can try every single possible key to check its correctness. This is time consuming, but feasible for several widely deployed ciphers.

An obvious way to conduct an exhaustive key search attack is to write software that will check each key in turn. Current microprocessors have clock rates in the gigahertz range and can execute several instructions per clock cycle. They are also cheap, highly available and easy to program.

Another possibility is to use a Field Programmable Gate Array (FPGA) device to conduct an exhaustive key search. FPGAs provide the functionality of a custom chip without the high up-front cost and lead time. They have much lower clock rates than general-purpose CPUs, but can be designed to perform one task exceptionally well. Parallelism can also be exploited to increase the overall search rate.

We thus have several questions requiring investigation with regard to FPGA technology in cryptanalysis:

- Which cryptanalytic tasks can FPGAs complete more quickly than CPUs?
- What are the price/performance benefits of FPGAs over CPUs?
- What other technologies are there that might allow us to complete these tasks faster or cheaper?

1.2 Why exhaustive key search?

Exhaustive key search is guaranteed to be a possible attack for any cipher, but not necessarily feasible. Most new ciphers that are being deployed have key lengths of 128 bits or greater. A cipher with such a key length cannot be feasibly attacked with current technology. Nevertheless, there are many reasons why conducting research into exhaustive key search attacks is worthwhile.

A lot of currently deployed encryption is vulnerable to key search attacks. The default encryption used by GSM mobile phones and 802.11b wireless networks uses a key which is short enough to facilitate exhaustive key search. The DES cipher was widely deployed in the banking industry (amongst others) and is vulnerable. Many websites using SSL encryption are also vulnerable.

Export restrictions in many areas prevent the use of ciphers with long key lengths. The United States has a history of restricting the export of strong cryptography, often using key length as a deciding factor. The Wassenaar agreement stipulates similar limitations and is enforced by 33 countries around the world, including Australia.

Small embedded devices may not be able to support ciphers with long key lengths. Cheap smart card devices containing encryption software are becoming more widespread. In order to meet cost or size constraints, many of these devices use very short key lengths or known weak ciphers. The encrypted data transmitted from many of these devices can be attacked using exhaustive key search.

Many other attacks employ an exhaustive key search. Many attacks work by reducing the key space to an amount which can be feasibly searched or by removing large sections of the key space that can be proven to not contain the target key. Time/memory tradeoff attacks usually require a large preprocessing step which resembles key search. Both of these attack types require a key search to be conducted as part of their operation.

Key search machines can be useful research tools. Research into other attacks may require a cipher to perform particular operations or to generate plaintext or ciphertext with certain characteristics. Exhaustive key search can be used to achieve this.

Weak encryption has been used extensively in the past. Significant amounts of information has been encrypted with ciphers that are vulnerable to exhaustive key search or other attacks. Encrypted data could be stored until the technology or techniques to reveal that data become available. Key search machines may still be able to reveal valuable information that was encrypted in the past. Similarly, future technology may be able to reveal even today's strongly encrypted data.

Exhaustive key search is highly parallelisable. This makes it a valuable application with which to experiment with parallel computing techniques.

1.3 Approach

In order to determine the utility of FPGAs when conducting exhaustive key search attacks, we need to consider their potential price and performance benefits over other technologies such as ASICs and CPUs. Pricing data can be obtained from suppliers, while performance data can be gathered from implementations. Performing implementations should also provide useful insights into the issues involved with cipher and key search machine design.

CPU pricing can be obtained from suppliers and performance measured with benchmark software. ASIC price and performance estimates can be obtained from suppliers.

The optimal family and device within each technology can be determined by computing the price for a certain search rate. Comparing price/performance ratios between technologies for different ciphers will help to determine which technology is best under what conditions.

From these analyses, it should be possible to recognise situations where FPGAs can be beneficial in key search applications.

1.4 Thesis organisation

Chapter 2 describes all of the past work, theory and knowledge that will be needed to understand the remainder of the thesis. It also sets the context for the new developments made by this thesis. Chapter 3 describes the design and implementation work that was performed in order to gather meaningful data. It allows the data analysis to use real-world data. Chapter 4 analyses the gathered data to form conclusions on a wide variety of areas, and forms the bulk of this thesis. Chapter 5 summarises the conclusions and provides directions for future work.

Chapter 2

Background

2.1 Theory

The theory in this section is only covered briefly. The reader is encouraged to refer to Bruce Schneier's *Applied Cryptography* [1] for more details.

2.1.1 Symmetric ciphers

A symmetric cipher is characterised by the functions

$ciphertext = E(plaintext, key)$

and

$plaintext = E^{-1}(ciphertext, key)$.

The intent of a cipher is that the function E be non-invertible without the key. This ensures that the plaintext remains secret to people without the key.

A *block cipher* is one where data is processed in discrete blocks. The input plaintext or ciphertext is broken up into blocks of the appropriate size. An example is DES, which processes data in 64 bit blocks. A *stream cipher* is one which works with much smaller units of data – often a single bit at a time. A5/1 is a common stream cipher. Stream ciphers are used to generate a *key stream*, which is then XORd with the plaintext to produce the ciphertext. XORing the ciphertext with the key stream again will decrypt the data.

2.1.2 Attacks and security

In a *known plaintext attack,* the attacker possesses some ciphertext and the matching plaintext. The goal is to find the key. This is the attack method usually used in research; possessing or being able to infer part of the plaintext is a reasonably safe assumption. E-mail headers and IP packets always begin in the same way, for example.

In a *ciphertext only attack*, the attacker possesses some ciphertext. These attacks are more difficult to perform. Usually, the attacker relies on some properties of the plaintext to determine when they are successful (such as character distributions or language statistics).

There are two criteria for a symmetric cipher to be considered secure [2]:

1. There must be no shortcuts to attack a cipher; exhaustive key search must be the most feasible attack

2. The number of possible keys is large enough to make an exhaustive key search attack infeasible

2.2 Key search attacks

In a key search attack, the attacker tries every possible key as input to the cipher. The known piece of ciphertext is also used as an input. In a known plaintext attack, the trial plaintext from the cipher output is compared to the known plaintext. In a ciphertext only attack, heuristics are used to determine if the output is valid plaintext.

2.2.1 Normal cipher usage

Most ciphers consist of a key setup phase and an operation phase. During key setup, the internal state is initialised. During operation, input ciphertext or plaintext is encrypted or decrypted. Key setup only needs to be conducted once for each key that is used.

When a cipher is used in practice, the key is usually kept constant for a long period while the plaintext or ciphertext input is varied frequently. Key setup is performed only once, and the cipher is designed to handle the rapid change in input.

Exhaustive key search reverses this by keeping the input constant while changing the key frequently. The main implication from this is that key setup must be performed very frequently. Many ciphers exploit this to improve resistance to key search attacks by having a very long key setup period. The key setup period is often comprised of the encryption algorithm itself. This greatly increases the time and resources needed to conduct a successful exhaustive key search.

Commercial chips that perform encryption or decryption may not be suitable for use in a key search machine if they are not designed to have the key changed frequently. Conversely, it may be possible to optimise a custom key search design by precomputing (partially evaluating) parts of the algorithm, since the ciphertext is known in advance. This technique has already been used to produce very fast and efficient cipher implementations by including the key in the design itself.

2.2.2 Block ciphers

When attacking a block cipher, one output block is usually tested for each key. If the output block matches the known plaintext, tests with more blocks are conducted to verify that the key is correct. The further checking step is important. There may be several keys that give the same plaintext output if the key size is longer than the block size and only a single block is checked.

2.2.3 Stream ciphers

Stream ciphers are often faster to conduct brute-force attacks against because incorrect keys can be quickly eliminated. A simple approach would be to generate a quantity of the key stream and XOR that with the ciphertext to generate the plaintext. The stream cipher can then be treated in exactly the same way as a block cipher. Efficiency can be slightly improved by ignoring the XOR stage and simply searching for the correct key stream. The amount of key stream to be generated must balance out the number of false alarms with the amount of time taken to check each key. Generating more of the key stream will cut down on false alarms, but take more time.

The main problem with this approach is that it is very inefficient. The entire block of key stream must be generated before it is checked for correctness. The first few bits to be generated may be enough to determine that a key is incorrect. A more efficient algorithm would then be:

1. Generate a single unit of the key stream (the smallest amount possible).
2. Check whether this unit matches the first unit of the desired key stream.
3. If it matches, continue checking with the next unit of the same key. If it doesn't, start again with the next key.
4. If a sufficient number of units match, return the key as a potentially correct key.

With this algorithm, an average of two units of key stream need to be generated for each trial key. This is far more efficient than the simple algorithm, which may need to generate a large amount of key stream to avoid returning an excessive number of potential keys.

As with a block cipher, generating the first unit of key stream may require a lengthy key setup phase be carried out. Many key search attacks avoid this by searching for the initial state of the cipher after the key setup has been completed. This is not always feasible; some stream ciphers such as RC4 have very large internal states.

Name	Key length	Type	Operations	Ref.
DES	56	Block	Bit permute, rotate, XOR, table lookup (6×4)	[3]
RC4	64	Stream	Add, table read and write (8×8), XOR	[1]
Rijndael	128/192/256	Block	Table lookup (8×8), rotate, multiply ($GF(2^8)$), XOR	[4]
3DES	112/168	Block	Bit permute, rotate, XOR, table lookup (6×4)	[1]
IDEA	128	Block	XOR, add, multiply (16 bits), rotate	[4]
Blowfish	32–448	Block	XOR, add, table read and write (8×32)	[5]
RC5	0–2040	Block	XOR, add, variable rotate	[6]
A5/1	64	Stream	XOR, shift	[7]
Skipjack	80	Block	XOR, shift, add, table lookup (8×8)	[8]

Table 2.1: Cipher operations

Table sizes are specified by (x×y), where x is the number of bits in the index and y is the number of bits in the output. For example, 6×4 can be modelled by a RAM with 6 address bits and 4 data bits. Addition is considered to include subtraction as a trivial extension.

2.3 Common ciphers

Most ciphers make use of a number of common operations. These operations typically retain entropy to ensure random-looking output. They may also introduce nonlinearities in the output. Ciphers generally operations from a number of algebraic groups to improve their strength.

By testing the relative speed of each operation on each implementation technology, it should be possible to gain insights into which ciphers will run quickly on which technology.

2.3.1 DES

The Data Encryption Standard (DES) has been extensively used and studied for decades. Several linear and differential attacks against it have been discovered, but the most effective attack remains exhaustive key search. It works with 64 bit blocks and was originally designed for fast hardware implementations. It has been the subject of several contests.

DES has enjoyed widespread military, government and commercial use in the past, notably within the banking and finance sectors. Its key length is only 56 bits, which is considered far too weak nowadays. Many attacks on the key length of DES have been performed, some of which are described below.

DES remains in use through a variant called Triple DES (or 3DES). In 3DES, the DES cipher is applied three times with two or three different keys. This is an effective method of increasing the strength of the DES cipher, but care must be taken during implementation to ensure that all of the keys are different. Clayton and Bond successfully attacked a secure

processor (formerly used in ATMs) that utilises 3DES [24]. They exploit protocol flaws to force the processor to use duplicate keys. They can then perform a key search attack on the reduced key space to determine a "master key".

2.3.2 RC5

RC5 is a simple block cipher designed by Ronald Rivest [6]. Despite its simple structure, very few attacks better than exhaustive key search have been discovered. It is fully parameterised, so the block length, number of rounds and key length can be selected to suit the application. It is best known as the cipher being attacked by the RSA Secret Key Challenges [29]. The parameters for the challenges are selected to be efficient on modern CPUs. RC5 is not widely used for commercial applications and has been patented by RSA Data Security.

2.4 Implementation technologies

2.4.1 Software on general-purpose CPUs

Software is usually the first platform that a cipher will be implemented with. General-purpose CPUs are cheap, plentiful, and fast for most tasks. Many ciphers are designed for an efficient software implementation.

2.4.1.1 Parallelism

CPUs are designed to perform serial operations very quickly. Regardless of the amount of available chip area, the need to operate serially remains. This has lead to modern CPUs expending a large amount of area on prediction and caching circuitry. A doubling in chip area for a CPU will not result in a doubling of performance.

Exhaustive key search is highly parallelisable; the task can be split perfectly between any number of processing units. This means that using multiple specialised processing units instead of a single CPU may give higher performance.

Recent CPUs have demonstrated a move towards increasing parallelism. Multiple execution units, deep pipelines, Symmetric Multiprocessing (SMP) and techniques such as Intel's HyperThreading all serve to increase the level of parallelism.

2.4.1.2 Bitslicing

Eli Biham pioneered a technique which later became known as bitslicing [9]. The paper deals primarily with its application to the DES cipher, but it is applicable to many other algorithms. In it, each register of a CPU is viewed as a large number of single-bit registers.

This allows a large number of single-bit operations to be performed in parallel. For an algorithm such as DES which is composed largely of single-bit operations, this provides a very large performance gain.

2.4.2 ASICs

An ASIC (Application Specific Integrated Circuit) is a chip that has been designed for a particular purpose. They are usually very fast for whatever application they have been designed for, but cannot be modified after fabrication. Initial fabrication costs are very high, but can be amortized over many chips. The unit price per chip is usually quite low. There is a far greater development effort required than for software, and more than for FPGAs. Gate array designs reduce the high cost of development significantly, but reduce achievable density. They work by placing gates over the entire silicon area of a device during fabrication and linking the gates with metal layers later.

FPGA designs can be converted to equivalent gate array ASIC designs at a relatively low cost. This technique was used for the machine described in [10]. Designs implemented in this way tend to be faster and cheaper than those on FPGAs, but not as fast as a dedicated ASIC design. It is an attractive option where development time and cost are important and the number of FPGAs needed make the implementation cost prohibitive. Many of the issues affecting FPGA designs (such as timing) also apply to ASICs. Routing tends to be much less problematic on an ASIC compared with an FPGA.

2.4.3 FPGAs

FPGAs (Field Programmable Gate Arrays) combine software and hardware approaches. They are chips that can have their internal layout reconfigured at any time. This is usually achieved by loading a *bitstream* from a ROM or controller. An FPGA typically contains a large number of logic blocks. "Programming" an FPGA determines how the logic blocks are wired together. Modern FPGAs may contain tens of thousands of logic blocks, each of which contains latches, combinational functions and other logic. High-end FPGAs such as the Virtex II Pro [11] even integrate CPU cores within the normal FPGA fabric. There is a move to providing dedicated hardware within FPGAs, such as RAM and communication controllers.

FPGA performance approaches that of an ASIC, but their general structure makes them slower and less efficient. Much less can be done within the same amount of silicon area. They also generate more heat and use more power than an equivalent ASIC. FPGAs do not have the high up-front cost of an ASIC, but cost more per unit. They are ideal for prototyping and development, since they can be reprogrammed quickly at no cost.

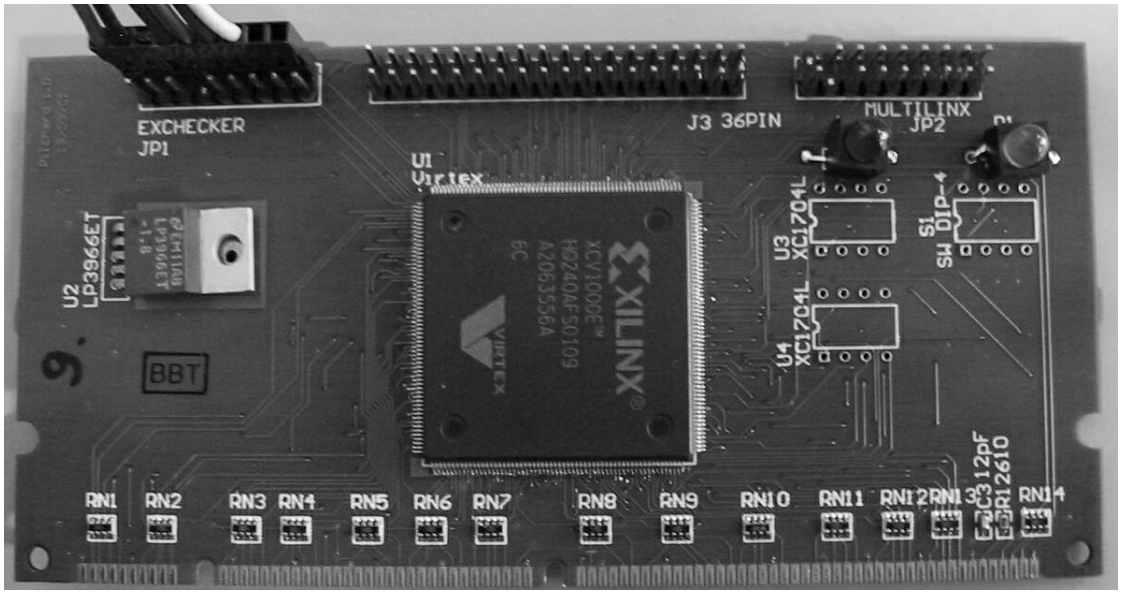

Figure 2.1: The Pilchard development board

Developing a design for an FPGA is generally more time consuming than writing normal software due to the low-level nature of the design. There are also far fewer competent FPGA designers than software programmers, increasing development cost and time.

2.4.3.1 Pilchard

Pilchard is a low-cost FPGA development board [12]. It contains a Xilinx Virtex E device; the device used for this thesis was an XCV1000E-6HQ240. The Pilchard board and FPGA device used are shown in Figure 2.1.

Pilchard interfaces to the RAM bus of certain motherboards. From the perspective of the FPGA designer, it provides a simple synchronous 100MHz or 133MHz bus with no interrupts or DMA. It appears as memory range to the programmer, so registers within the FPGA can map directly to variables in the driver software. This combination allows easy system development, high performance and low FPGA resource requirements.

A number of external I/O pins are available that can be interfaced to other hardware. Space is also available on the circuit board for ROM chips that can load a bitstream into the FPGA device on power up.

2.4.4 Combining technologies

Software, FPGA and ASIC components can be profitably combined. All known FPGA and ASIC-based key search machines are controlled by a general purpose computer.

One useful idea is to use each technology for the task it excels at. For example, a machine using all three technologies might use a computer as a primary controller, FPGAs as lower-level controllers, and ASICs for each search unit. The computer is useful for human interface and easy reconfigurability. The FPGAs are useful for their high speed, I/O capabilities and reconfigurability. ASICs have the advantages of very high speed and low price in very large quantities. This scheme is particularly desirable for ciphers where suitable ASICs are available commercially, reducing fabrication costs.

Another possibility is to perform hardware-fast operations on ASICs and FPGAs, and software-fast operations on computers. This is generally infeasible due to the "I/O gap" – latencies between the ASIC/FPGA and the CPU far outweigh any speed benefit. Pilchard interfaces with the memory bus of a computer and can thus provide a very high bandwidth and low latency connection to the CPU. The integrated CPUs in Virtex II Pro FPGAs also provide similar benefits.

The integrated CPUs on Virtex II Pro FPGAs provide another option for ciphers favouring software implementations. Each Virtex II Pro FPGA contains up to four PowerPC 405 cores [11]. These CPUs could be used to perform the bulk of the cipher operations while the surrounding FPGA fabric handles control, communication and testing of results. A very large number of CPUs could be integrated into a small space using this technique.

2.5 Previous work

Most previous hardware key search machines have been designed to locate DES keys. This is because DES is very fast in hardware, widely deployed and has a dangerously short key length. There are also political issues involved with DES and its selection as a standard.

2.5.1 Hardware key search machines

Many hardware key search machines have been produced in the past. Most of these are not practical machines. They are used to gather performance estimates with a certain technology or technique. More key search machines are known to exist; only those with notable features have been presented in this section.

Most of these performance figures have caveats; they may be estimates, approximations, or based on implementations which were not completely carried out.

2.5.1.1 Performance estimates

A number of papers provide theoretical estimates of the cost of breaking ciphers with a hardware key search engine. *Minimal Key Lengths for Symmetric Ciphers to Provide Adequate*

Name	Cipher	Year	Level	Technology	Keys/sec/chip	Ref.
Diffie/Hellman	DES	1977	Theoretical	ASIC	1M	[13]
McLaughlin	DES	1992	Theoretical	ASIC	2k	[14]
Wiener	DES	1993	Designed	ASIC	50M	[15]
Goldberg/Wagner	DES	1996	Built	CPLD	0.5M	[16]
Various	DES	1996	Estimated	FPGA	30M	[2]
Various	DES	1996	Estimated	ASIC	200M	[2]
Wiener	DES	1997	Estimated	ASIC	300M	[17]
Kaps/Paar	DES	1998	Built	FPGA	6.29M	[18]
EFF	DES	1998	Built	ASIC	60M	[10]
Hamer/Chow	DES	1999	Built	FPGA	25M	[19]
Goldberg/Wagner	RC4	1996	Built	CPLD	8.4k	[16]
Kundarewich/Wilton/Hu	RC4	1999	Built	CPLD	40k	[20]
Tsoi/Lee/Leong	RC4	2002	Built	FPGA	6.06M	[21]
Goldberg/Wagner	A5	1996	Built	CPLD	4M	[16]

Table 2.2: Previous hardware key search machines

Commercial Security [2] is a prime example of this. It lacks practical grounding, but is still contains useful estimates and background. In 1996, it predicts that a $200 FPGA (AT&T ORCA) can test 30 million DES keys per second, and that a $10 ASIC can test 200 million DES keys per second. For $300,000, an FPGA-based machine should be able to crack a DES key every 19 days, and an ASIC-based machine every three hours.

Wiener's *Efficient DES Key Search* [15] describes a theoretical hardware DES key search machine based around a custom ASIC. The machine was designed, but not built. Just about every detail of the machine was described, including the schematics, interfaces and physical requirements. Each ASIC in the design is estimated to be able to check 50 million keys per second. Pipelined search units and an LFSR key generator are used. In 1993, a machine costing $100,000 is estimated to be able to crack a DES key every 35 hours, on average. These estimates were updated in 1997 to take newer technology and further analysis into account [17]. In this paper, a $100,000 machine should be capable of cracking a DES key in six hours. The speed estimates given in this paper are the basis of those presented in [2].

McLaughlin presented a high-level design for a DES key search machine [14]. The paper ignores low-level issues and focuses on the high-level functionality of the machine. Its main features are the use of a fuzzy comparer and specialist key generators.

Diffie and Hellman produced a paper in 1977 that counters objections to the possibility of a key search machine [13]. Objections to the reliability, size, speed, power and cost of a key search machine were countered, and a system architecture based around a million search chips presented. Despite the (comparatively) primitive technology available at the time, a key search machine is still believed to be feasible.

2.5.1.2 The EFF DES Cracker

In 1998, the Electronic Frontier Foundation (EFF) published a book [10] describing a large scale DES cracker that they built. Paul Kocher later elaborated on the book in [22].

The machine was based around a very large number of search units. Each search unit takes 16 clock cycles to check a DES key. 24 search units were built into a custom ASIC design that ran at 40MHz. 64 ASICs were placed on each circuit board and 27 boards constructed. Taking into account faulty search units, the entire machine was capable of a search rate of 92.6 billion keys per second, or an average search time of 4.5 days. The machine was built with a budget of $250,000.

A flexible plaintext recognition scheme was used that allows selective matching against certain characters, as well as specialised modes for the Blaze Challenge [23]. This allowed the machine to conduct ciphertext-only attacks.

Kocher further elaborated on the technical problems inherent with building such a large machine. Power and heat issues were the main ones dealt with. The ASICs used had to be produced successfully with a single attempt, leading to a number of design compromises. Had this requirement been lifted, both the performance and correctness of the design could have been improved significantly.

Many of the political issues involved with DES were also covered. These focused primarily on the disparity between what government officials report and what the DES cracking machine is capable of.

2.5.1.3 Small-scale key search machines

Many small key search engines have been produced in an attempt to gauge how processing power has changed with time. These are all based on reconfigurable hardware (FPGAs or CPLDs).

Hamer and Chow implemented a DES key search machine on the Transmogrifier 2a, a system containing 32 linked Altera FPGAs [19]. Their design features a long DES pipeline and an LFSR key generator design that minimises the need for key schedule logic. Each FPGA ran at 25MHz, giving an aggregate search rate of 800Mkeys/sec.

Tsoi, Lee and Leong implemented an RC4 key search machine [21] using a Pilchard board. Their design used 96 search units running at 50MHz to achieve a total rate of 6.06Mkeys/sec. Not all of the FPGA resources were used; the number of search units was limited by the number of RAM blocks available. The FPGA implementation ran approximately 58 times faster than a software implementation running on a Pentium 4 1500MHz. Kundarewich, Wilton and Hu also implemented an RC4 key search machine using Altera CPLDs, and obtained a search rate of 40kkeys/sec [20]. Brief cost and performance comparisons were

carried out.

Deeper investigation into architectural decisions was made by Kaps and Paar [18]. They explored the idea of an algorithm independent key search machine on an FPGA, focusing on DES. Several architectural options for DES were investigated and implemented on Xilinx FPGAs. Their key search design would be capable of 6.29Mkeys/sec.

Goldberg and Wagner performed an analysis of RC4, A5, DES and CDMF implementations on a CPLD board [16]. The performance of a variety of CPLDs was compared with their cost, noting that low end CPLDs generally give the best price/performance ratio. Comparisons were also made against equivalent software implementations. The RC4 cipher was found to be faster in software than hardware, the opposite result to that of [21] and [20].

2.5.1.4 Specialised key search machines

Clayton and Bond exploited a variety of protocol flaws to successfully attack a security module that was previously used in ATMs [24]. They were able to successfully recover 3DES keys from the device with the assistance of a cheap FPGA board. By implementing a more practical attack they were able to learn more about the difficulties and benefits of working within a real environment.

Pornin and Stern attacked A5/1 using a combination of software and hardware approaches [25]. Software was used to reduce the search space of initial states, while hardware was used to conduct an exhaustive search over this subspace. A board containing four Xilinx 4010E FPGAs was used in conjunction with a 500MHz Alpha workstation. Each FPGA contains 12 search units, each checking one initial state every 65 cycles. The FPGAs were clocked at 50MHz, giving a total search rate of 37 million initial states per second. Using two boards with one workstation allowed an initial state to be determined in 2.5 days on average, far faster than exhaustive key search alone. Keller and Seitz took a more analytical approach by using backtracking to reduce the search space [7]. Their implementation was performed on a Xilinx XC4062 FPGA.

2.5.2 Software key search

2.5.2.1 Distributed computing

Several organisations have implemented software to conduct distributed key search attacks against ciphers using network-connected hosts. distributed.net [26] is the largest and most well-known of these. Others include DESCHALL [27] and SolNET [28]. The basic idea is the same: run a piece of software on many hosts and coordinate their efforts with a central server. The software is configured to run during idle time on the hosts. Buffering schemes allow hosts to continue working on their part of the task when not connected to a network.

Regardless of the precise task being performed, work is usually divided into "blocks" which take a (relatively) short period of time to complete. A client connects to the server to be allocated a number of blocks and does not communicate again until those blocks are complete.

These efforts have been quite successful so far. distributed.net has successfully completed RSA Data Security's RC5-64, RC5-56, DES II-1 challenges [29], as well as a similar challenge from CS Communications & Systems [30]. They completed the DES-III challenge with the help of the EFF DES cracker. DESCHALL completed the DES-I challenge. A group headed by Germano Caronni and containing 3500 computers completed the RC5-48 challenge. Ian Goldberg used 250 computers to complete RC5-40 [31].

Chapter 3

Design

3.1 Conceptual design

Most key search machines are designed around similar ideas. A controller operates a number of independent search units. This controller usually interfaces with a general purpose computer. Each search unit contains a key generator, decryptor (or encryptor) and comparator. The key generator produces trial keys that need to be checked. Some designs combine the key generator and decryptor modules to improve performance. The decryptor decrypts the known ciphertext with the trial key. An encryptor can also be used with some limitations. The comparator checks the plaintext that is generated by the decryptor to see if it is correct. If it is, the controller is signalled.

Due to its complexity, the cipher module is usually considered to be the bottleneck in the system. All other modules must be able to operate at least as quickly.

3.1.1 Key generator

A counter is an obvious choice for a key generator. The count sequence is predictable, but performance is not adequate on all devices. Xilinx FPGAs provide a dedicated carry chain which improves performance significantly.

The EFF DES cracker [10] uses a counter where the 24 most significant bits are held constant and the 32 least significant bits counted. This technique is useful to reduce a counter's propagation delay. The most significant bits must be counted and loaded externally. This scheme introduces the idea of a "block" of keys – a subset of the key space which can be searched in a short period of time. The 24 constant bits can be viewed as the block number. There must be a mechanism with which the controller can detect the end of block condition and start the key generator on a new block.

One design [12] uses a single counter that is shared between all of the available search

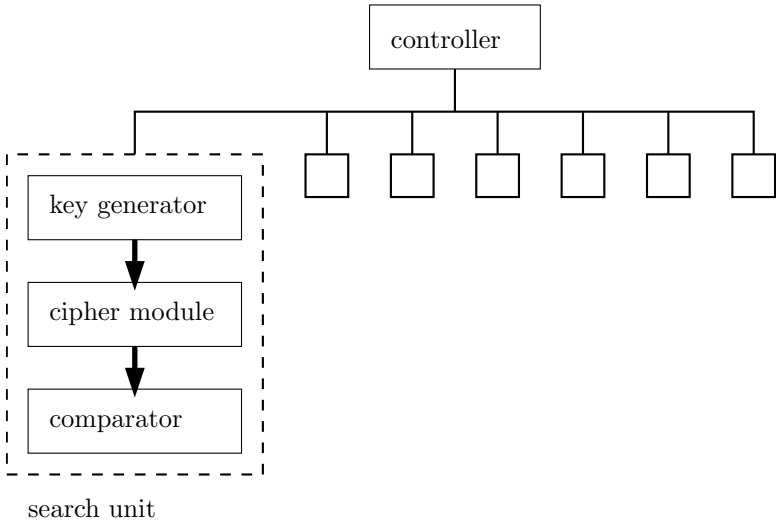

Figure 3.1: Conceptual key search machine design

units. Each unit adds or concatenates a unique ID to the counter value to obtain its trial key. This scheme works well when the number of search units is a power of two since the ID can simply be concatenated, saving resources.

A number of designs [16, 19, 15] use Linear Feedback Shift Registers (LFSRs) to generate trial keys. The main advantage of an LFSR over a counter is its high speed; propagation delays remain constant regardless of the length of the LFSR. One disadvantage of LFSRs is that their count sequence is nonlinear. Evenly breaking up a large key space between search units requires more effort than with a linear counter. One simple scheme is to use a shorter LFSR than usual and set the remainder of the key bits to a constant value. This works similarly to the block scheme for linear counters described above; the LFSR can be 32 bits long, and the remaining 24 bits set by the controller.

3.1.2 Cipher module

3.1.2.1 Encryptor vs. decryptor

When performing a known plaintext attack, the choice of encryptor or decryptor is dependent on which has higher performance. Most ciphers (including all stream ciphers) have identical performance regardless of their mode. Some may have a more efficient implementation when implemented in one way or another. RC5 [6] is an example of this. An RC5 encryptor can operate more efficiently than a decryptor because the order that the S array is used in during key setup matches that used in the encryption stage, allowing the phases to overlap.

Most key search machines will use a decryptor. Ciphertext-only attacks require a decryptor. Known-plaintext attacks will also require a decryptor under some conditions. This

is to allow a more flexible comparator scheme that can detect correct plaintext regardless of imperfect knowledge.

3.1.2.2 Iterated vs. pipelined

Two major approaches are used when implementing the cipher module; a long pipeline or a small iterative module. Most ciphers are comprised of a number of round functions, making an iterative implementation natural. Pipelined approaches can achieve much greater speeds at the expense of FPGA resources. DES is frequently implemented as either a small iterated module or a long pipeline. The iterated version takes a multiple of 16 cycles (one for each application of the round function) to produce one block of output, while the pipelined version can produce one unit of output every clock cycle. The resource gains made by using an iterated cipher implementation almost never outweigh the loss of speed. Resource constraints may force a cipher to be implemented in iterative form.

An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists [32] explores these issues in depth. It presents FPGA performance figures for the MARS, RC6, Rijndael, Serpent and Twofish ciphers. Loop unrolling, pipelining and sub-pipelining are investigated as architectural choices. In most cases, a pipelined implementation was fastest. *Fast DES Implementation for FPGAs and Its Application to a Universal Key-Search Machine* [18] explores pipelined, combinatorial and iterative approaches for the DES cipher.

3.1.2.3 Partially evaluated circuits

Some ciphers may benefit from having values precomputed during compilation time. This is usually used to achieve higher performance in systems that have very infrequent key changes. FPGAs fit well with this approach, allowing the programmed-in key to be changed with only a small period of downtime. The speed efficiency of an FPGA DES implementation was improved significantly using this technique [33]. The utility of this technique in a key search machine is dependent on the cipher. The plaintext or ciphertext would be compiled into the design instead of the key. This may yield improvements for some ciphers, but exploratory experiments only showed very small resource savings.

3.1.3 Comparator

The environment that the key search machine operates in determines the choice of comparator. If a perfect ciphertext/plaintext pair is known, simply checking for bit equality will be adequate. Ignoring certain bits in the trial plaintext may be a useful extension when only a portion of the sought plaintext is known.

If a ciphertext-only attack will be attempted or the plaintext is not precisely known, it may be necessary to implement a heuristic matching scheme. Such a scheme will generally flag a number of keys as potential matches and allow humans or software to check them further for correctness.

A simple scheme to detect ASCII text is to require that the most significant bit of each plaintext byte be 0. This can be further generalised into a statistical approach that scores each plaintext byte in the plaintext according to its probability of occurrence. *A Programmable Plaintext Recognizer* [34] uses similar ideas to extend Wiener's theoretical key search machine [15]. Applying compression to a message before encrypting it causes their heuristics to fail. This is an effective countermeasure against any statistical comparator, since the compression makes the message "look like" random data.

Some applications may also benefit from a specialist comparator. A machine designed to solve the Blaze Challenge [23] would need a specialist comparator that will find a match on any block that fits the form of the solution (in this case, when the plaintext is composed only of a single repeated byte.)

3.1.4 Returning matches

At some point in a key search machine's operation it will be necessary to return potential keys to the host computer. Several schemes have been used to achieve this goal.

Most key search machines simply stop running when a match is found and wait for the computer to read out the key value. This is simple and flexible, but inefficient when many keys need to be returned – while a search unit is waiting to release the key it halted on, it cannot be used to search the key space.

A hardware buffer can be used to reduce the waiting time. When a key needs to be returned it is read into the hardware buffer, and the controller can read the keys out. This has the advantage of improved efficiency, but costs hardware resources.

One novel approach is to measure the amount of time needed to find the key. Using knowledge of how quickly the key space can be searched, an approximate trial key can be found. A number of keys need to be checked to account for timer inaccuracies. This method removes the need for key storage and retrieval hardware.

3.2 A generic FPGA key search machine

The programming interface for this design is supplied in Chapter A.

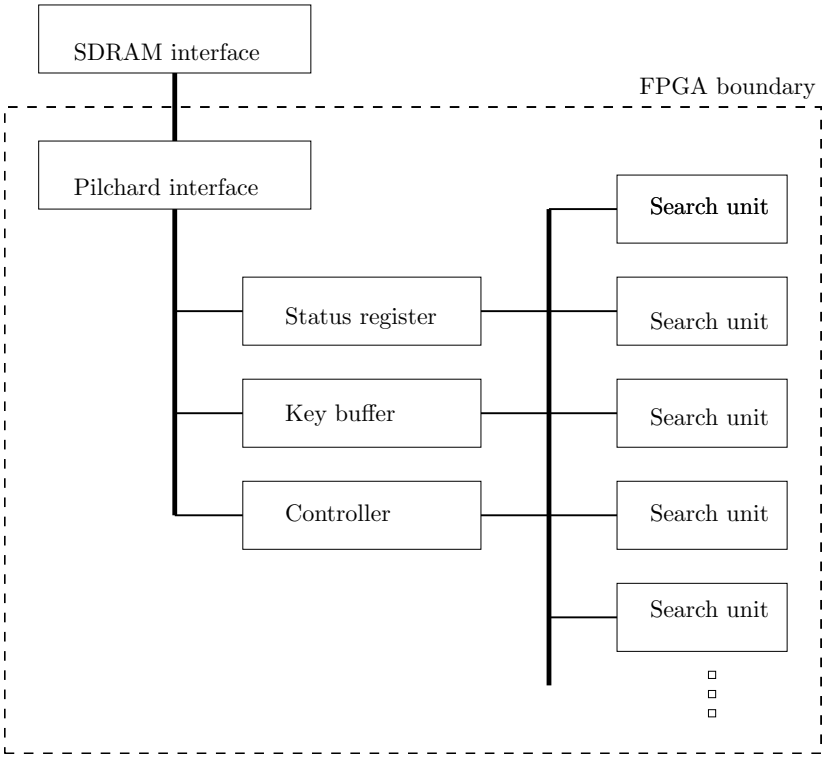

Figure 3.2: Initial key search machine top level design

3.2.1 Goals

To produce a FPGA-based key search machine which can operate independently of cipher algorithm. It should communicate with a computer for instructions and data. It should be reasonably scalable for large key cracks, and be easily modifiable for ciphertext-only attacks. It should allow rapid prototyping of key search machines for different ciphers.

3.2.2 Top-level design

The bus provided by the Pilchard interface runs synchronously at 100 or 133MHz. The remainder of the key search machine must operate at this speed.

The top-level Status register provides general status information for the entire key search machine.

The key buffer stores potentially correct keys for the computer to read out and check further. This prevents search units from being paused for very long when a potential key is located. It is particularly useful for ciphertext-only attacks, where there may be a large number of potentially correct keys. 256 keys can be stored; this figure fully utilises the four Block SelectRAM units that are needed to store a 64 bit word.

The controller operates the search bus. It relays commands from the computer to indi-

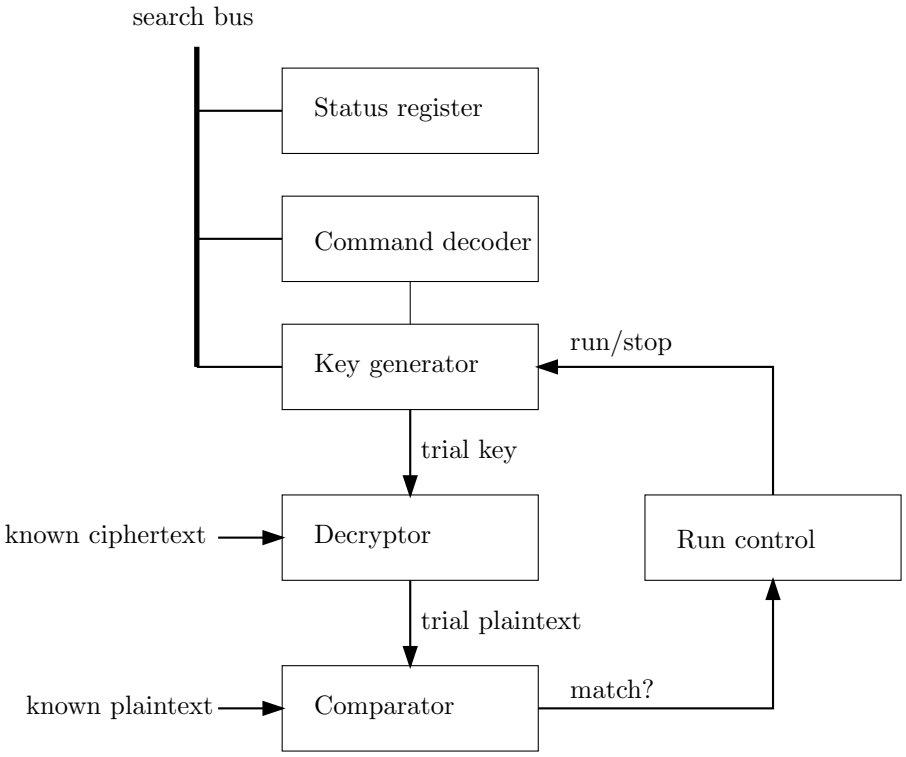

Figure 3.3: Initial key search machine search unit design

vidual search units, stores the ciphertext and plaintext registers, and polls each search unit on the bus to see if there are any keys waiting. It uses a simple state machine. While there are no commands waiting to execute, it polls search units to see if there are any keys waiting. If a key is found, it is read into the key buffer. If a command arrives, it temporarily stops polling and executes the command.

3.2.3 Search unit design

Each search unit has its own status register which the controller uses to determine if a key has been located. The key generator provides a trial key to the decryptor, which uses the key and the supplied ciphertext to produce trial plaintext. This trial plaintext is compared with the known plaintext or has a set of heuristics applied to determine if it appears to be valid. If it is, the search unit is halted until it is instructed to restart by the controller.

3.3 Another generic FPGA key search machine

3.3.1 Motivation

Several problems were identified with the original key search machine that justified the design of a new one.

- It was overly complicated. Modules within the design such as the key buffer provided very low returns on their cost. Removing the key buffer allowed the controller to be simplified, since it no longer needed to poll search units for keys. Other simplifications were made within the search units.
- Timing closure was never achieved. The original design was implemented before a proper understanding of high-speed digital logic had been attained. The fact that it actually worked can be put down to luck and favourable operating conditions.
- Minor bugs remained that complicated the software design.
- The programming interface was more sophisticated than it needed to be, which complicated the software further.
- The clock speed for search units was locked to that of the memory bus. This turned out to be a larger handicap than was originally predicted. The DES search unit could run at almost 180MHz according to the synthesis tools, but was still locked to the 100MHz of the memory bus. This could be increased to 133MHz by adjusting the motherboard jumpers, but this was not a satisfactory solution. Better support for slow ciphers was also needed.
- It could not easily support key lengths over 64 bits.

The new design and its driver software took approximately four days to implement and debug. The programming interface is described in Chapter B.

3.3.2 Design

There are two controllers in this design; a master and a slave. The master handles all communication with the host computer and links to the slave with an asynchronous bus modelled closely on VME. The slave controller's only purpose is to link the asynchronous bus and the search bus, which can run synchronously at any speed. This allows search units to run at any speed, simplifying cipher implementation.

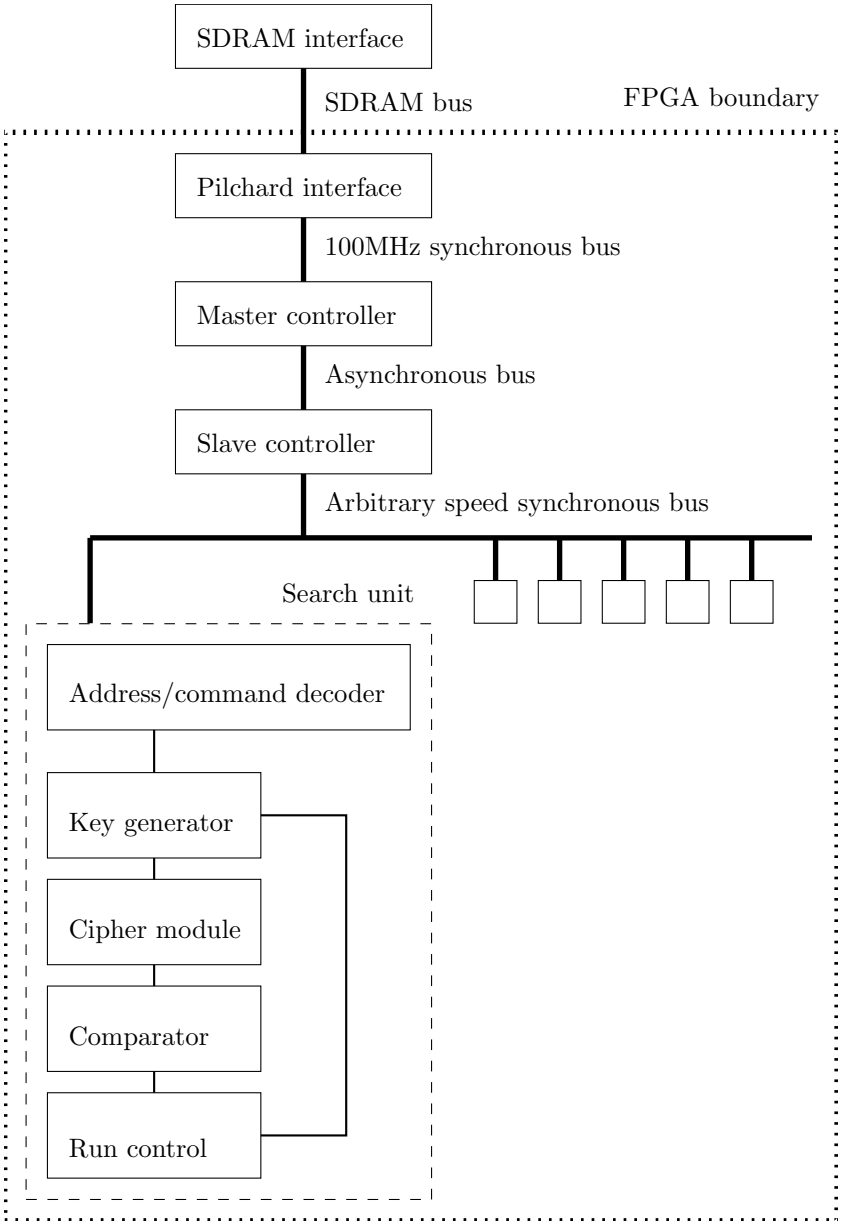

Figure 3.4: Revised key search machine design

3.3.3 Search unit design

The search unit is designed to use a block system for key allocation. Only the block number is transmitted to the search unit. When retrieving the key from the search unit, the least significant 32 bits of the key are returned. The software is expected to track which search unit is searching which block.

3.3.4 Search bus

The search bus runs at the same speed as the search units. The two clock domains (SDRAM clock and search unit clock) are linked with an asynchronous bus using similar protocols to VME.

High speed search units were later identified as a problem; it was found to be difficult to route a wide high speed bus over the entire FPGA and still meet timing constraints. A potential improvement to the machine would be to decouple the clock rate of the search bus from that of the search units or make the bus completely asynchronous. The latter was the original intent of the asynchronous bus, but the resources required to implement it made it unwieldy to use on every search unit. It remains the best solution for a large-scale machine (at least between FPGA devices).

3.4 Modules

3.4.1 Counters

Two linear counters were implemented. The first was a simple 64 bit counter. The second was designed to work with block schemes and added functionality to allow counting to be inhibited for ciphers that do not need a new key every clock cycle. It counts through 32 bits of range and has a further 48 bits of range that is set externally.

3.4.2 Bit equality comparator

A comparator that checks for exact bit equality was implemented. It was 64 bits wide. It flags a match when its two inputs are identical. To ensure that it runs quickly enough with high clock speeds, it was implemented as a short pipeline. On the first clock cycle, four 16 bit segments of the trial plaintext are compared individually. On the second cycle if the result of these four comparisons is true, a match is flagged.

3.4.3 Statistical comparator

A simple statistical comparator was implemented using some of the ideas within [34]. Its purpose is to use the probabilities of different bytes within the produced plaintext to determine if the plaintext "looks right". The definition of "looks right" varies depending on the attack scenario; English text would have different statistical properties to an executable file, for example.

The algorithm used is fairly simple. The comparator takes a 64 bit input and splits it into 8 bit bytes. Each byte value has an assigned "score" – higher scores correspond with more frequently occurring byte values. The scores for each byte are added and compared against a threshold value. If the threshold is exceeded, a match is flagged.

3.4.3.1 Implementation

Implementation of the algorithm was more challenging, but still straightforward. The main design constraint was that the comparator be no slower than any decryption module – in this case, the DES module running at over 149MHz and producing one word of plaintext per cycle. In order to meet this timing requirement, steps of the algorithm were split up as much as possible.

Figure 3.5 shows the steps performed by the implementation. Four RAM blocks were used to store the byte value scores (8 bits each). Each RAM block has two ports, allowing a total of eight memory lookups every cycle. The scores are added in parallel in pairs to minimise delays on each cycle. Finally, the total score is compared with the threshold (which is set by the plaintext value). Splitting up the steps in this way produces a deep pipeline, but allows very high clock rates.

The threshold comparison stage is the main timing bottleneck. Speed improvements can be made by reducing the comparison resolution. By only comparing the most significant four bits, synthesis reports a maximum speed of 181MHz. The required resolution depends on the statistical properties of the text being attacked.

A small C program was written to generate character scores from files. It counts the frequency of each character in the file. The scores are then normalised down to 8 bits and output in a format suitable for entry directly into the VHDL RAM initialisation code. This data could also be used to modify the bitstream after compilation if desired. This program allows the comparator to be "trained" on similar input data to what is expected.

3.4.4 DES cipher module

The DES implementation is a modified copy of the DES demonstration provided with the Pilchard board, which is itself a modified version of the Xilinx optimised DES implementation

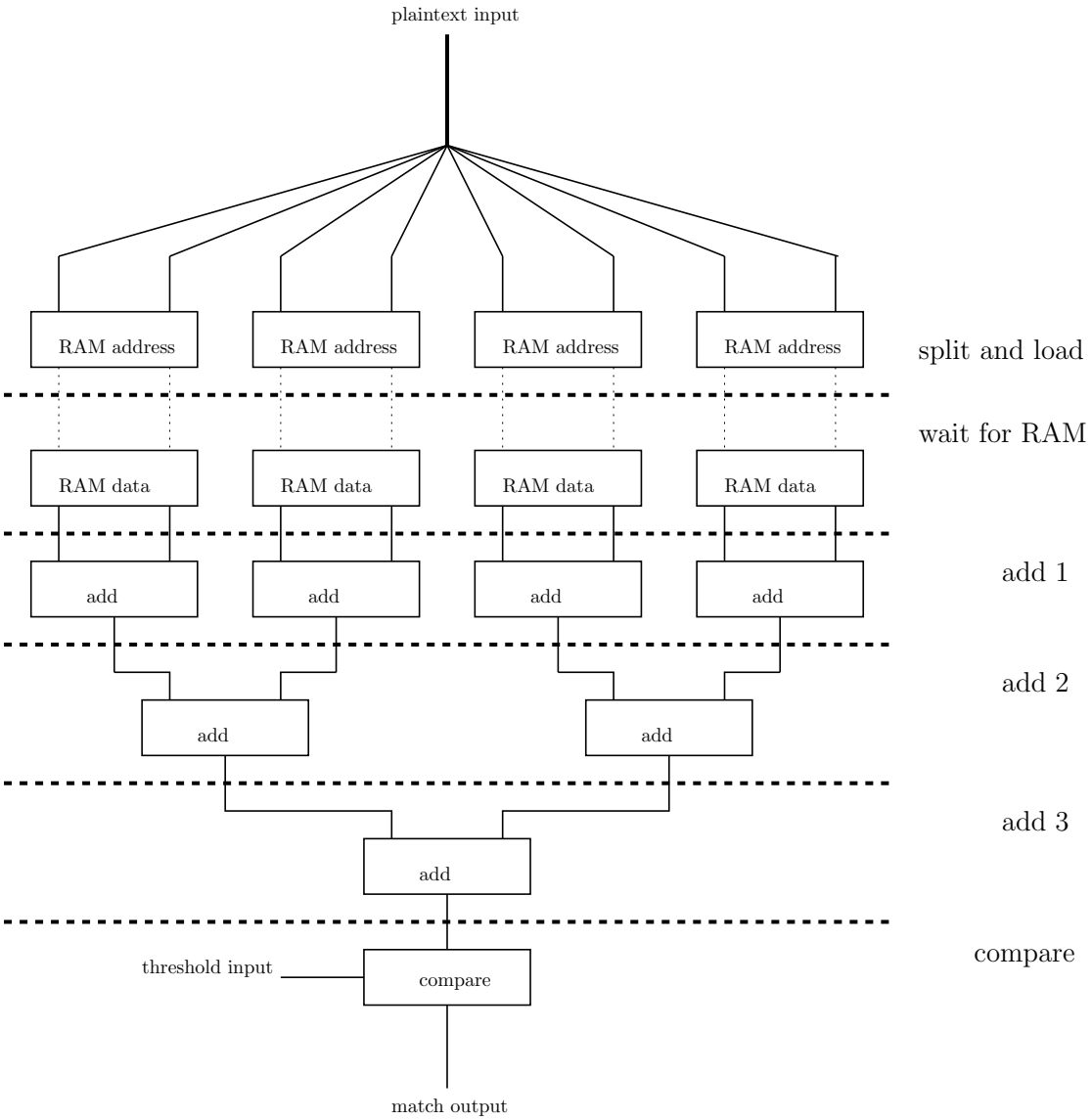

Figure 3.5: Statistical comparator design

in [35]. The order of the round functions was reversed to convert the encryptor into a decryptor.

Registers were added to the key schedule logic, but later removed when the efficient keying system described in [19] was implemented. This scheme integrated an LFSR key generator with the DES key schedule logic. A 72 bit LFSR with taps suitable for a 56 bit LFSR was used. As the previous keys were shifted through the LFSR they remain available to the key schedule logic, which can generate the necessary subkeys with rotations. This saved approximately 500 slices that were previously used for subkey registers. Subkey generation was essentially free, although fanout on the LFSR bits did reduce the speed slightly.

The possibility of attacking a key and its complement simultaneously was considered. This halves the search space, but not the search time. The decryption portion of DES comprised the bulk of the area requirement in a hardware implementation, and this improvement only saves key schedule logic. After implementing the LFSR keying scheme above, performance improvements would be negligible.

Using the XCV1000E, a key search machine containing controllers and five search units was operated at 100MHz, giving a total search rate of 500Mkeys/sec.

3.4.5 A5/1 cipher module

The A5/1 implementation was produced from scratch using the algorithm description given in [7]. It aims to find the initial key state rather than the key itself. Time constraints did not allow the more efficient stream cipher attack in Section 2.2.3 to be implemented, and so no further work was performed using this module.

The (already small) resources needed to implement the A5/1 module could be further reduced by configuring the Xilinx LUTs as shift registers [36]. This would complicate key loading; the entire key state could no longer be loaded in a single cycle.

3.4.6 RC5 cipher module

3.4.6.1 Introduction

The RC5 implementation was produced completely from scratch using the algorithm description given by Rivest [6]. It implemented RC5-32/12/9. It was intended to be used to complete the RSA Secret-Key Challenge contests [29]. The possibility of connecting the complete key search machine to distributed.net [26] was considered as an extension.

Few prior works in this area could be located. [37] claims to have schematics for a functional RC5 implementation on Xilinx FPGAs, but they are no longer available. The author was not able to be contacted. [38] contains a Verilog model which was not found to be useful.

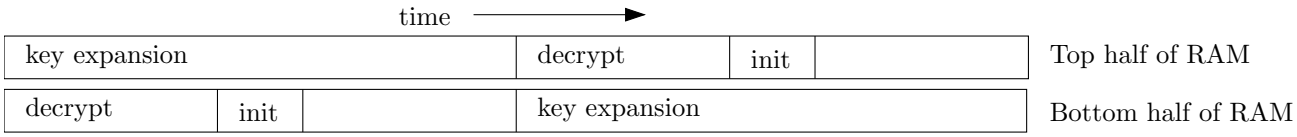

Figure 3.6: RC5 RAM timing

3.4.6.2 Pipelined design

A fully pipelined design similar to that used for DES was investigated. This possibility was considered to be impractical due to the large number of registers needed for the S array.

After implementing the iterative version, the possibility of implementing a pipelined version was considered again. This time, the number of LUTs required was identified as being excessive. A prototype implementation determined that each stage of the key mixing phase would require 256 LUTs, and each half-round of the decryption phase would require 192 LUTs. Given 78 mixing steps and 24 decryption half-rounds, the number of LUTs required is 24576 – coincidentally, the exact number of LUTs available on the Virtex 1000E. Many more would be required for state decoding, communication, key generation, comparisons, routing overhead and so on. This possibility was not investigated further, but would almost certainly be feasible given more hardware resources to work with. Such an implementation would be able to provide very high search rates on sufficiently large FPGA devices.

3.4.6.3 Iterative design

An iterative design for the RC5 implementation was used. Block SelectRAM memories within the FPGA were used to store the S array. The number of RAM blocks was anticipated to be the limiting factor, similar to the RC4 key search engine described in [21]. The L array was stored in three rotating registers; this eased timing constraints and prevented reads and writes to the RAM becoming a bottleneck.

The key mixing phase of RC5 took the bulk of the time needed to check a key. It required 78 iterations, each of which consists of a read and a write to the S and L arrays. To minimise the time required per cycle, the key mixing stage of the algorithm was set up to operate continuously on two separate regions of RAM. The initialisation and decryption stages were arranged to work on the opposite region of RAM. When a key mix phase completes, the decryption and initialisation phases begin on that region of RAM. In this way, the average time required to check a key would effectively be the time required to perform the key mixing phase.

The key mix phase needs to be completed as quickly as possible. The decryption and initialisation phases are not timing critical, and can be completed more slowly in order to

save FPGA resources. The decryption module takes advantage of this by performing twice as many rounds and interchanging the A and B registers at the end of each round. In this way the subtract, shift, XOR and RAM lookup resources can be reused. The initialisation module actually performs the additions required to initialise the S array, even though these results could be trivially precomputed. This saves FPGA resources.

The general goal for the key mix operation is to complete as quickly as possible. The general goal for the decryption and initialisation operations is to use as few resources as possible, so long as the time taken for these two operations does not exceed that needed by the key mix operation.

3.4.6.4 Implementation

One problematic area in the implementation was the 32 bit barrel shifter required by RC5. The initial naïve implementation required 352 slices; with the help of [39] this was improved to 80 slices. One shifter is required for each of the key mix stage and the decryption stages. These account for a significant amount of the resource usage. Some research and experimentation was conducted to find smaller or faster shifter designs, without success. Shrinking or speeding up the barrel shifters would provide large benefits to the overall performance of the design.

Running the module at 100MHz proved difficult. Routing delays introduced after the place and route stage were the cause of the problem; congestion was present at one of the RAM blocks. The delay at this point increased when the number of search units was increased, suggesting that floorplanning may be useful to reduce the delay or at least make it consistent. A brief unsuccessful attempt at floorplanning was made.

To solve this problem, two approaches were used. Originally, two RAM blocks were used to provide a 32 bit wide RAM. One port was used by the key schedule module and the other by the decryptor and initialisation module. The number of RAM blocks was doubled and writes made to both pairs. Reads could be made from either pair of RAM blocks, allowing unrelated logic to be moved to different areas of the FPGA by the place and route tools. This helped to reduce delays. The RAM blocks were not being otherwise used. Adding a wait state after RAM access allowed the module to meet its timing requirements at the cost of reduced performance.

The total time required to check an RC5 key is 469 clock cycles. Each iteration needs 6 clock cycles, and 78 iterations are required. One cycle is needed for initialisation. At the target clock speed of 100MHz, this gives a search rate of 213,220 keys/sec. 16 search units could be fit into an XCV1000E device, giving an aggregate search rate of 3.4Mkeys/sec.

The RC5 cipher module consumed 595 slices. The implementation in [37] required 510 XC4000 CLBs; each XC4000 CLB [40] is roughly equivalent to a Virtex slice.

3.4.6.5 Optimisations

The possibility of increasing the clock speed of the RC5 module was investigated, but found to be counterproductive. The intent was to balance the time spent in each pipeline stage better, hopefully overcoming the increase in resource usage and number of stages required. Registers were inserted at locations responsible for timing limitations. These registers did not increase resource usage significantly due to the structure of the Virtex slice [41]. The number of cycles per round increased from 5 to 8 and the synthesis clock speed from 102MHz to 142MHz, which was not an effective tradeoff. Many previously trivial operations such as the comparison needed to be split into stages instead of being simple combinatorial operations, which greatly increased the complexity of the source code. The overall resource usage also increased.

Replacing each bit in the three registers used to implement the L array with a short LUT shift register would reduce the resources allocated and potentially ease routing.

Some work was conducted to see if it was possible to take shortcuts in the key mixing operation; this was unsuccessful.

Including the ciphertext and IV at synthesis time reduced resource usage for the search unit to 539 slices. This would be a worthwhile approach for an attack where the ciphertext and IV are known in advance. It would generally not be suitable for an ASIC implementation.

This module was implemented before the second key search machine. Performance could be improved by running at a lower clock speed with fewer pipeline stages.

3.5 Software benchmarks

3.5.1 Methodology

Benchmarks were conducted on a number of different CPUs to measure how quickly they could perform key searches. Setting up and running the benchmarks was very rapid, so many different CPUs were tested to determine if any would provide significant price/performance advantages.

Pre-written benchmarks were used. These benchmarks were faster and more thoroughly tested than what could otherwise be produced in the available time.

3.5.2 Results

Each benchmark was run at least three times until consistent results were achieved. Linux benchmarks were run as the root user, prefixing the benchmark command with “nice -20” to ensure that the benchmark ran with the highest priority.

Tables containing the gathered results are given in Chapter C. distributed.net maintains an online database [42] of search rates for each CPU, allowing some of the benchmark results to be verified.

3.5.2.1 DES

Two benchmark programs were used: the distributed.net client version 19991117 (which had to be compiled from source), and the SolNET DES client [43]. The distributed.net client gave far better benchmark results, but could only be run on Linux machines with appropriate compiler versions. Neither DES client had been optimised for modern CPUs.

The command "dnetc -benchmark des" was used to run the distributed.net benchmarks, and "desclient-x86-linux -m" for the SolNET benchmarks. The SolNET client's benchmark results were unstable on faster CPUs, requiring them to be run a large number of times.

distributed.net maintains an online database of search rates for each CPU [42]. The DES benchmarks for newer CPUs could not be verified because the CPUs did not exist at the time that the online benchmarks were gathered. The results for older CPUs were far higher than those in the online database.

Benchmarks for Celeron, P4HT and Athlon XP (Barton) CPUs had to be inferred from others based on the same core. The Mkeys/sec/MHz ratios obtained for RC5 remained fairly constant under this assumption, and this is assumed to remain true for DES.

3.5.2.2 RC5-72

The distributed.net client version 03033120 was used to conduct RC5-72 benchmarks. Binaries from the distributed.net website were downloaded for the relevant platform, unpacked, and the benchmark executed from the command line with "dnetc -benchmark rc5-72".

The RC5 benchmark results were verified against those in the distributed.net database. Confusion is apparent with the Athlon speed ratings; it is not obvious whether an entry marked "1900" refers to a 1900+ or a 1900MHz Athlon. Nevertheless, the RC5 benchmark results gathered were found to mesh well with those in the database.

No Celeron machines based on the Pentium IV core were available to run benchmarks on, so the online benchmark results were used for analysis. These appeared internally consistent, so a Mkeys/sec/MHz rating was determined and averaged across the available benchmark results to reduce error. This rating was used to infer the missing benchmark results.

Chapter 4

Analysis

4.1 Factors affecting exhaustive key search

Many factors affect the time taken to conduct an exhaustive key search attack:

- **Key length.** Increasing the key length used by a cipher will dramatically increase the size of the key space and hence the time required. Using long keys is by far the most effective countermeasure to an exhaustive key search attack.
- **Available resources.** An attacker with more resources (money, computational power, people) can sweep the key space more rapidly.
- **Cipher design.** The design of a cipher has a strong influence on how long an exhaustive key search will take. Several factors contribute to this:
  - **Key setup time.** In normal use, a cipher's key setup time is usually negligible. When conducting a key search attack, the key setup phase must be performed for every key. Long key setup times can frustrate key search attacks.
  - **Cipher operations.** Some operations will take longer to perform than others. Some ciphers are designed to use operations that are fast on one particular technology and slow on another.
  - **Cipher speed.** On identical implementation technologies, some ciphers can encrypt or decrypt more quickly than others.
  - **Frequency of register access.** Pipelined designs can achieve significant resource savings if registers are accessed infrequently (discussed in Section 4.3.) Iterative designs often have less constraints on their RAM or register access.

- **Availability of accurate plaintext and ciphertext pairs.** Possessing only ciphertext or incomplete ciphertext and plaintext may reduce the speed and accuracy of the exhaustive key search attack. Ideally, several perfect pairs of ciphertext and plaintext should be available. If many pairs are available, time-memory tradeoff attacks may be more appropriate for some ciphers.

4.1.1 Software factors

The most significant factor that affects the time required to conduct a key search using CPUs is the word length of the cipher. The speed of the cipher tends to be highest on a CPU whose word length matches that of the cipher. All processing must be performed in units of this word length.

Ciphers that have very small states may be able to operate completely within CPU registers, which will improve performance. Smaller speed benefits will arise if the state is small enough to fit inside the highest-level cache. Very few ciphers have state sizes over a few thousand bytes.

Some cipher implementations may exchange storage space for processing power by precomputing values. One software implementation of A5/1 [44] exploits this by precomputing all possible values of the individual LFSRs. Instead of performing the normal shift and XOR operations, output is generated by modifying pointers to the precomputed values. This requires almost 128MB of memory, but greatly improves the performance of A5/1 in software.

4.1.2 FPGA factors

The most significant factor affecting the speed of exhaustive key search on FPGAs is whether the cipher can be implemented as a long pipeline as opposed to an iterative structure. Pipelined structures make far more efficient use of the FPGA resources and can achieve higher clock speeds. Any cipher can be implemented as a long pipeline, but the required FPGA resources are often prohibitive. Generally, a long pipeline will check one key every clock cycle.

4.2 Operation performance

4.2.1 FPGAs

Operation performance on FPGAs is influenced by the time required to complete the operation and the resource usage. Many operations can be completed more quickly by using

Operation	LUTs	Time (LUT depth)
Bit permute	0	0
Fixed rotate or shift	0	0
XOR (up to 4 inputs)	1	1
Data-dependant table read or write (1–4 address bits)	1	1
Data-dependant table lookup (w address bits, $w > 4$)	$2^{w-3} - 1$	$w - 3$
Add or subtract (two inputs)	1	Depends on data width
Variable rotate or shift over w bits	$\log_2 w$	$\frac{\log_2 w}{2}$

Table 4.1: LUTs and time required per bit for cipher operations

more resources. Conversely, using less resources allows a greater level of parallelism, increasing overall performance. Finding the correct balance is difficult and may require several attempts at implementation.

Results giving the time and space requirements of a number of operations are given in Table 4.1. Not all operations are covered, and optimisations using RAM or other FPGA features are ignored. In particular, large data-dependent table lookups will be more efficiently performed using the RAM blocks contained within most Xilinx FPGAs. Similarly, multiplications may be better performed using onboard Virtex II, Virtex II Pro or Spartan 3 multiplier resources.

Addition and subtraction are dependent on the width of the data being added, due to the Xilinx carry chain. Variable rotation is achieved using the scheme in [39], and assumes the use of Xilinx slice multiplexers. Data-dependent table lookup does not assume this because each Xilinx family has different multiplexer resources available which will affect the structures used for implementation.

4.2.2 CPUs

In contrast with FPGAs, storage space is not a concern for software implementations. When the word size of the operation inputs is equal to or less than that of the CPU, most modern CPUs can complete any operation very quickly. The operations being performed thus become less important.

Word size is the major factor when determining operation speed on CPUs. If the word size of the operation is greater than that of the CPU, performance will generally be halved (or slightly worse).

If the word size of the operation is less than that of the CPU, processing resources are being wasted. We then need to examine the algorithm to determine if any parallelism can be applied to make the best use of the resources that are available. For example, it may be possible to pack a number of short-word operands into one word and perform an operation

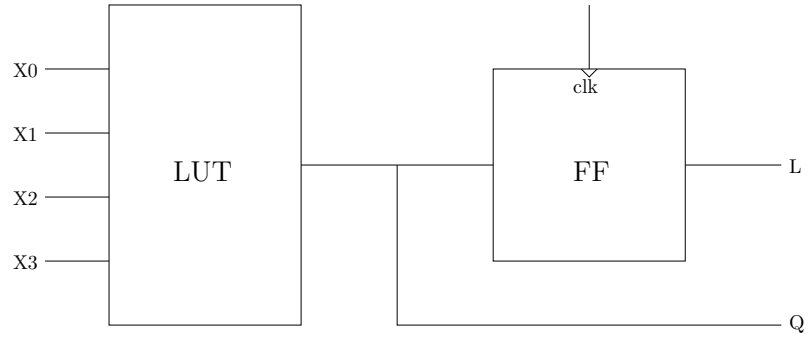

Figure 4.1: Simplified FPGA logic cell

over a number of words simultaneously. Bitslicing [9] is another approach that is used to accelerate single-bit operations on CPUs with a wide native word length.

4.3 Estimating FPGA resource usage for pipelined cipher implementations

4.3.1 Introduction

It is useful to determine whether a pipelined cipher implementation is feasible before beginning work on the implementation itself. This section describes a method that can be used to estimate the resource usage of a cipher given details of its algorithm.

4.3.2 Assumptions

A number of assumptions are made to facilitate this analysis:

- FPGAs are comprised of many logic cells (LCs.) A logic cell is comprised of a four input look-up table (LUT) and a flip-flop (FF). This is similar to the LC layout used in most Xilinx and Altera FPGAs, and is shown in 4.1.
- The state size remains constant throughout the key setup and decryption phases. This is rarely true, but does not significantly affect the results. The analysis can be extended to handle changing state sizes.
- The key setup and decryption phases are composed of a number of rounds. Each round performs the same operations. The logic used to perform each round maps the state from round r to round $r + 1$.

s	State size in bits
n	Number of rounds needed to complete a phase
r	Number of LUTs required to perform a round
m	Number of bits of state modified during a round
c	Number of LCs required to perform a phase

Table 4.2: FPGA resource estimation variables

4.3.3 Factors

To estimate the quantity of resources required, we consider:

- The size of the state during key setup and decryption
- The operations that are performed during key setup and decryption
- What quantity of the state is modified during each step or round of key setup and decryption

The final result will specify the number of LCs required for a fully pipelined implementation of the cipher. The mapping from LC to real FPGA resources is generally trivial. A Virtex, Spartan IIE, Virtex II or Virtex II Pro slice maps to two LCs. Spartan 3 devices have fewer effective LCs because half of the LUTs on the device have reduced functionality and do not support usage as a RAM or a shift register.

4.3.4 Method

We define a number of variables, shown in Table 4.2.

The state size s is the number of bits that need to be stored between rounds. This includes all registers and modifiable lookup tables used in a phase. Usually this figure can be determined by summing the total number of bits of storage used by the cipher.

The number of rounds n is usually specified by the cipher. Modelling the cipher in this way forces each round to complete in one clock cycle, which may result in very long combinational delays for some ciphers. This will be discussed further below.

The number of LUTs r required to perform a round is determined by examining at the operations performed during that round and the number of bits that these operations are performed on. These results are summarised in Table 4.1. Another strategy would be to implement the round function and use synthesis results to estimate the resource usage.

If a state bit is modified during a round, its result can be stored in the same LC as the LUT that modified it. A dedicated LC is needed otherwise. This gives us value m.

The number of LCs needed to complete a cipher phase is thus:

$$c = n(r + s - m) \tag{4.1}$$

We can then compare the number of LCs in the result to the number of LCs in a device to determine whether a pipelined implementation is feasible. Alternatively, we can use the LC estimate to determine what the smallest device needed is. The LC figure can also be used as a "difficulty rating"; ciphers with high LC counts will generally take more FPGA resources and time to attack.

4.3.5 Optimisations

4.3.5.1 Multiplexers

Each Xilinx slice contains a number of additional multiplexers which can reduce the number of LCs needed for large table lookups.

4.3.5.2 Shift registers

Xilinx LUTs can also operate as shift registers [36]. This is very useful when a state bit is not modified for one or more clock cycles. Instead of chaining FFs together, a single LUT can replace up to 16 FFs. Analysis of the data dependencies in the cipher algorithm can provide justification to reduce the effective s value significantly. Synthesis tools will sometimes perform this optimisation automatically. We can then obtain an alternate equation to obtain the number of LCs required:

$$c = S + nr \tag{4.2}$$

where S is the number of LCs used to store the state over the entire pipeline. Implementations using shift registers for storage cannot pack the shift registers into the same LC as the LUT performing the calculation, since the shift register uses the LUT. All state that is modified can be latched in the same LC as the LUT that performed the calculation. We thus do not require variable m.

4.3.5.3 Short pipeline stages

Pipelined implementations on FPGAs can be sped up by making each pipeline stage as short as possible. No additional cost is incurred when using the latches included in LCs, making very deep pipelines at high speeds a good design approach. Instead of completing an entire round in a single clock cycle, it is usually more efficient to break up the round and complete it over more clock cycles (at a higher clock rate). The design will still check an average of one key per clock cycle at the higher clock rate.

Splitting up the pipeline stages requires analysis of the data dependencies within each round, and would significantly complicate this analysis.

Using short pipeline stages will increase the number of LUTs that will need to be used as shift registers, and hence increase overall resource usage slightly. At worst, doubling the clock rate of a design will double the number of LCs used only for data storage. Smaller penalties will be incurred if shift registers are less than half full.

Some operations (particularly boolean operations like XOR) can often be collapsed into other operations, particularly if there are spare LUT input lines. This is highly dependent on the cipher algorithm.

4.3.6 RC5 example

Rivest describes the RC5 algorithm in [6]. It consists of three phases: initialisation, key mixing and decryption. The initialisation phase is ignored because it can be trivially pre-computed and will not consume FPGA resources when implemented in this way.

4.3.6.1 Key mixing

The key mixing phase of RC5-32/12/9 uses an S array of 26 words and an L array of 3 words. Each word is 32 bits wide, giving a total state size (s) of 928 bits. 78 rounds (n) are needed. Each round modifies 64 bits of state (m.)

To determine r, we examine the operations being performed. All of the table lookups operate in a predictable order, removing the need for additional multiplexers. Each round contains five adds, a fixed rotation and a variable rotation, all over 32 bits. One of the adds ($A + B$ in the second line) is performed twice, and can be ignored. This gives an r value of $32(4 \times 1 + 0 + \log_2 32) = 288$ and a final c value of $78(288 + 928 - 64) = 89856$. This represents a slice count that can only be achieved in very large FPGAs. The estimated r value is close to the value of 256 obtained in the trial implementation (Section 3.4.6.2.)

Significant resource savings can be achieved by recognising that each bit in the S array is only accessed once every 26 rounds, and each bit in the L array is only accessed once every 3 rounds. We can thus collapse a significant number of the LCs used purely for storage into shift register LUTs. 7488 LCs are used to store the L array as it travels through the pipeline; this can be reduced to 2496. 64896 total FFs are used to store the S array. Two shift registers are needed to provide the 25 cycle delay on each bit, and each bit is accessed three times. The total number of LCs needed is thus $26 \times 32 \times 2 \times 3 = 4992$. This demonstrates that the frequency of register access has a significant bearing on the efficiency of cipher implementations on FPGAs. The S array stores almost 9 times as much data as the L array, but requires only twice the resources.

Using Equation (4.2), we obtain an LC count of 22464 for the key mixing phase of RC5.

The resource usage can be further optimised by noticing that elements of the S array are initialised sequentially, and so not all values need to be stored until 26 rounds have been completed.

4.3.6.2 Decryption

The decryption phase of RC5 uses the same S array as the key mixing phase, but does not require the L array. Each round consists of two half-rounds which are identical except for the source and destination of the results. We can use this property to double the number of rounds and halve the number of LCs required per round. It consists of 22 half-rounds, each of which contains a subtraction, a variable rotation and an XOR. All operations are performed on 32 bit words. Two additional subtractions are performed after the half-rounds. We thus have $s = 26 \times 32 = 832$, $n = 22$, $r = 32(1 + 5 + 1) = 192$ (which matches the trial implementation) and $m = 64$. This gives $c = 22(192 + 832 - 64) = 21120$.

Again, significant resource savings can be achieved by using shift registers for storage. Each word in the S array is only accessed once and is never written to, so we need only provide storage for the period between the start of the decryption phase and the point where it is accessed. At worst, this will be 26 rounds, requiring two shift registers per bit. There are 26 words storing 32 bits each, giving an LC count of $26 \times 32 \times 2 = 1664$. Applying Equation (4.2) again gives $c = 1664 + 24 \times 192 = 6272$. The final subtractions require an additional 64 LCs, giving a total of 6336 LCs.

4.3.6.3 Results

Using the figures determined from Equation (4.1), we obtain a total LC count of 110976. This translates to 55488 slices – barely fitting within the largest Virtex II Pro device that is planned for production (and is not yet even shipping.)

Using the Xilinx-optimised figures from Equation 4.2 gives a total LC count of 28800. This converts to a far more practical 14400 slices, within the capacities of many larger devices.

It should be noted that the figures generated by this analysis technique tend to be conservative and ignore many potential resource optimisations. It also ignores issues of pipelining within the cipher rounds which are difficult to deal with in such a general sense. Both of these areas can provide significant resource and speed advantages in an actual implementation.

4.4 FPGA price/performance comparison

Pricing data for Virtex E, Spartan IIE, Virtex II and Virtex II Pro FPGAs in quantities of 25-99 was obtained from Avnet's website [45], and is current as of 15 October, 2003. The XC2V40 and XC2V80 device pricing is for a quantity of 100 or more.

Pricing data for Spartan 3 devices was obtained from Ernest Peltzer [46] of Sensory Networks, and are projected prices for Q1 2004 in quantities of 100 or more. Spartan 3 devices only started shipping very recently and so pricing data is both difficult to obtain and very likely to change.

This analysis assumes that the cipher module is the slowest part of the total key search machine. It ignores resources that would be dedicated to the PC interface, but includes those that are required for each search unit. Interface overheads are ignored because in a large-scale design each FPGA does not need its own PC interface and controller; the search bus can be connected directly to FPGA pads.

4.4.1 Speed grades and packaging

Each FPGA is available in a variety of speed grades and several packaging options. In general, the slowest speed grade and the smallest package gives the best price/performance ratio. Low FPGA speeds can be compensated for by using more FPGAs, and packaging is not important since only a few I/O lines are needed. This greatly simplifies the analysis by allowing a large percentage of the FPGA devices available to be ignored.

4.4.2 Families

The maximum attainable speed and resource usage is the same within each FPGA family. This allows performance estimates to be generated with far less effort. Synthesis estimates for the DES (Section 3.4.4) and RC5 (Section 3.4.6) search units are used. These are less accurate than those obtained after place and route, but remain valid across different capacity FPGAs within a family. These results are summarised in Table D.1. Search unit resource costs are estimated in the same way.

The maximum clock speed for a given design varies greatly depending on the FPGA family used. Interestingly, the current "budget" family (Spartan 3) achieves the highest clock rates. This can be explained by their 90nm manufacturing process; the Virtex II and Virtex II Pro use 150nm and 120nm processes.

RC5 requires half as many RAM blocks on a Virtex II, Virtex II Pro or Spartan 3 as it does on a Virtex E or Spartan IIE. This is because the RC5 implementation needs a 32 bit wide RAM, and the Virtex E and Spartan IIE RAM is only 16 bits wide.

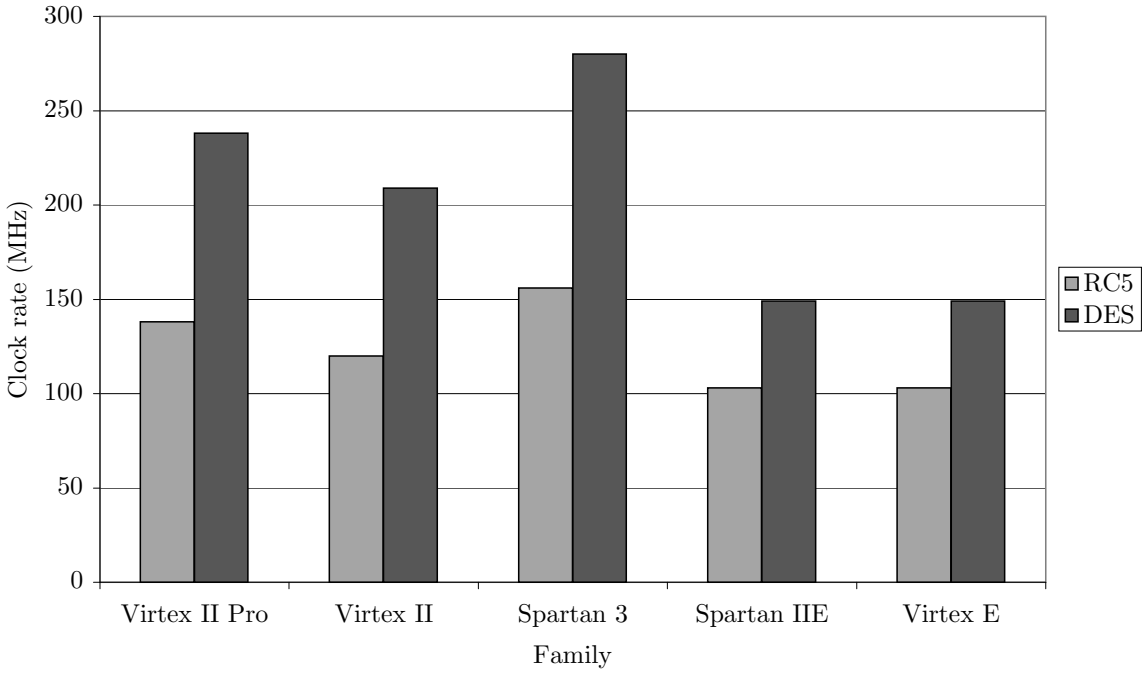

Figure 4.2: Relative clock speed across FPGA families

4.4.3 DES

Figure 4.3 shows the price and performance of each device in each Xilinx FPGA family for the DES cipher. Figure 4.4 shows the same data, but zoomed to show detail for the low-end FPGAs. "Kinks" in the graph appear where moving to the next largest device does not improve performance (since there are not enough available resources to add another search unit). Smaller kinks are visible when moving to the next largest package.

We can see that Spartan 3 FPGAs give by far the best performance for a given price. This is not surprising; they are positioned as a budget FPGA and can achieve higher clock rates than the other families being considered. Again, it should be pointed out that they have only recently started shipping and pricing will likely be volatile for some time. The pricing figures used for the analysis were also projected figures for Q1 2004 and for a larger quantity than the other families.

Of the mature families, Spartan IIE devices give the best price/performance ratio. Their performance is quite limited, however. Virtex II Pro devices can achieve spectacularly high search rates, but at a high cost per device.

Figure 4.5 shows the price/performance ratio achieved by each device in the Virtex II Pro family. The device with the best ratio is the XC2VP20, with the XC2VP30 and XC2VP40 close behind. In a real system where PCB costs and assembly have to be taken into account, it may be worthwhile purchasing a smaller number of faster FPGAs with a worse

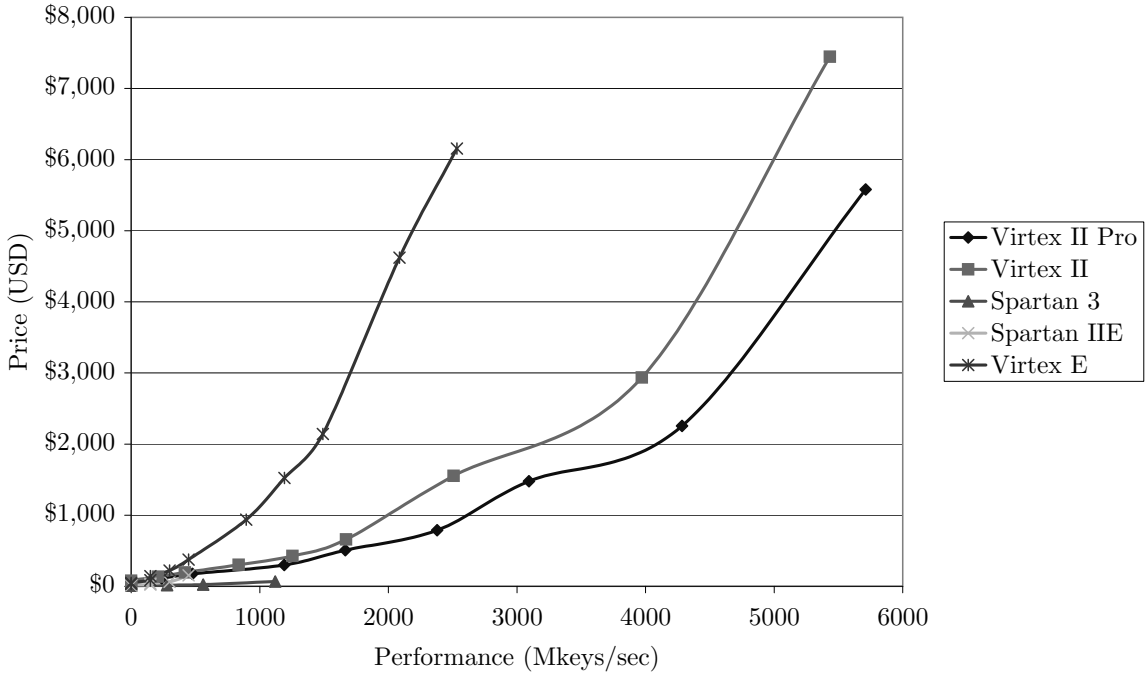

Figure 4.3: FPGA price/performance for DES

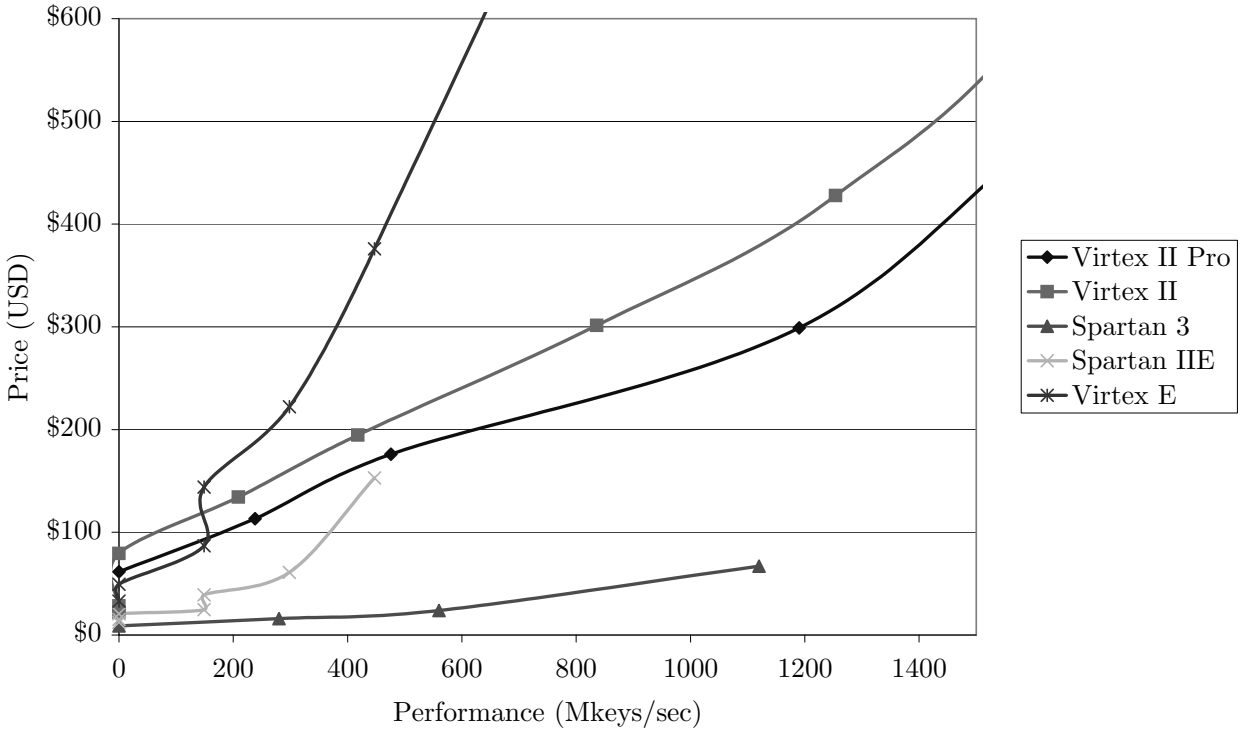

Figure 4.4: FPGA price/performance for DES, showing low-end detail

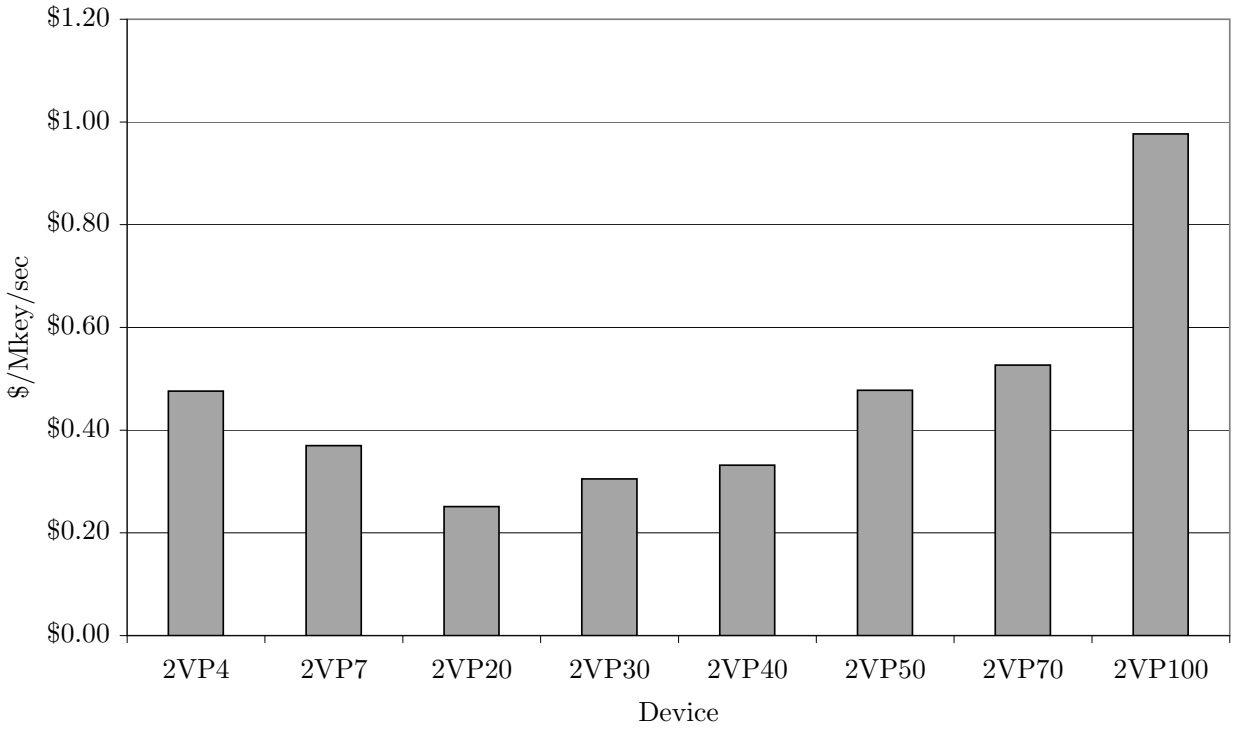

Figure 4.5: Virtex II Pro price/performance ratio by device for DES

price/performance ratio. These three devices use the FG676 package; the jump in price to the XC2VP50 can be explained by the larger package (FF1152). The XC2VP100 has a far worse ratio than the others, and a much larger package (FF1696). Like the Spartan 3 devices, it has only recently started shipping and may still have unstable pricing.

4.4.4 RC5

Relative FPGA price and performance for RC5 is shown in Figure 4.6. It is similar to that for DES, but with less gap between the Virtex E and Virtex II/Virtex II Pro families. Detail near the low end is also very similar. Relative pricing and performance within the family remains the same as for DES.

4.4.5 RC4

RC4's performance is entirely constrained by the number of RAM blocks available on the FPGA. This gives quite difference price/performance results. From Figure 4.7, we can see that the Virtex E and Virtex II families are quite similarly placed. Virtex II Pro devices perform much better when cost is taken into account. Examining the low-end detail (Figure 4.8) shows that the Spartan IIE family remains competitive for far longer than the Spartan

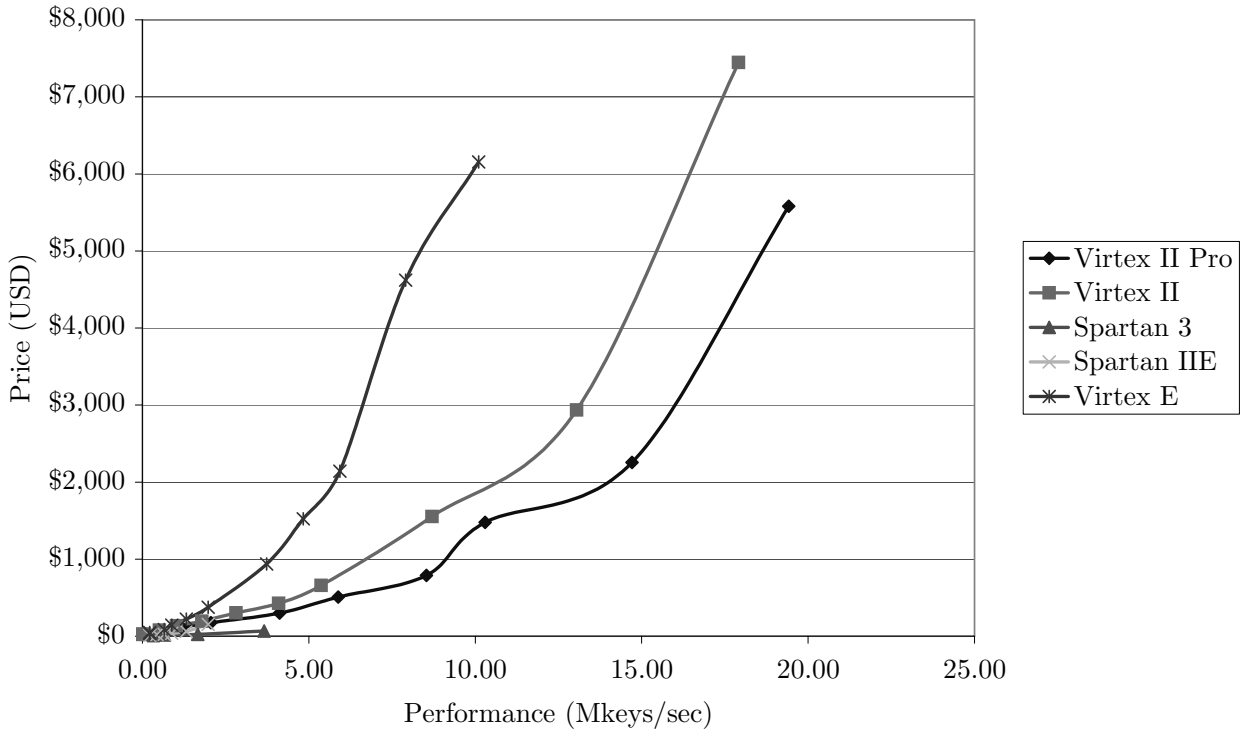

Figure 4.6: FPGA price/performance for RC5

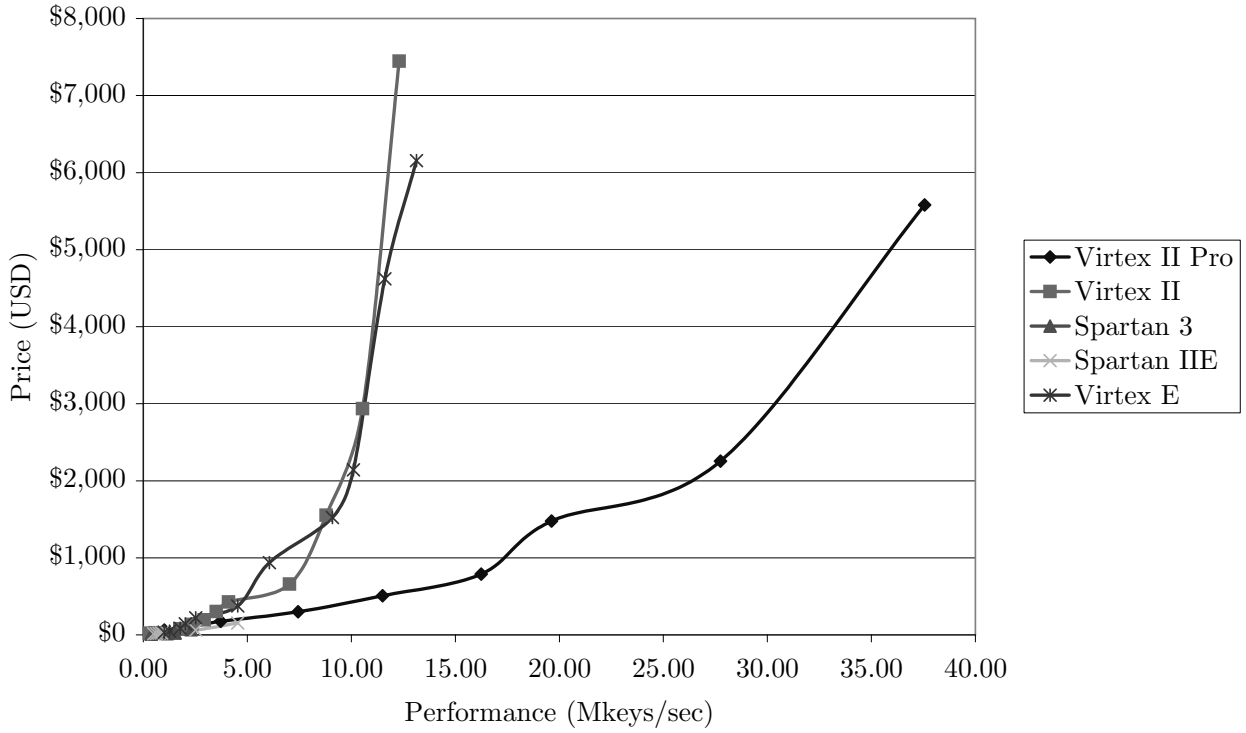

Figure 4.7: FPGA price/performance for RC4

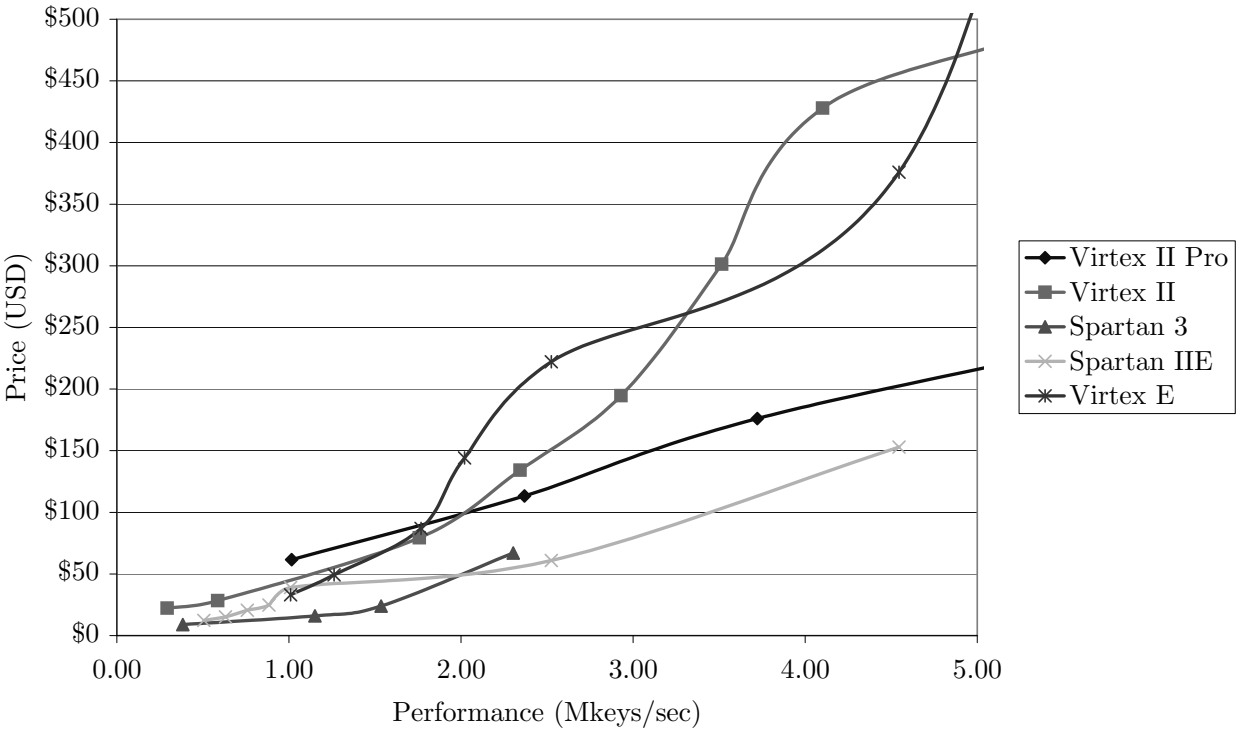

Figure 4.8: FPGA price/performance for RC4, showing low-end detail

3, in contrast with the other results. Again, the XC2VP20 remains the most cost-effective choice in the Virtex II Pro family.

4.5 CPU price/performance comparison

CPU pricing data was obtained from Sastradi Satria of OnLine Centre [47]. This is for quantities of 10 and is specified in AUD without GST. This is useful for comparing CPUs, but makes comparing price/performance ratios between CPUs and FPGAs difficult. Several other pricing sources were located but not used due to accuracy or quantity issues.

Benchmark results are listed in Section C, and should be interpreted taking into account the problems noted in Section 3.5.2. These results were scaled by the clock speed to obtain performance estimates for each CPU that is currently being sold. This assumes that performance will scale linearly with CPU speed, which is generally true for exhaustive key search.

When comparing performance between CPUs the SolNET benchmarks were used because they have been performed on a wider variety of CPUs. They are not directly applicable to CPU to FPGA comparisons.

No pricing data was available for mobile Intel CPUs (PIII-M, P4-M, Centrino). This

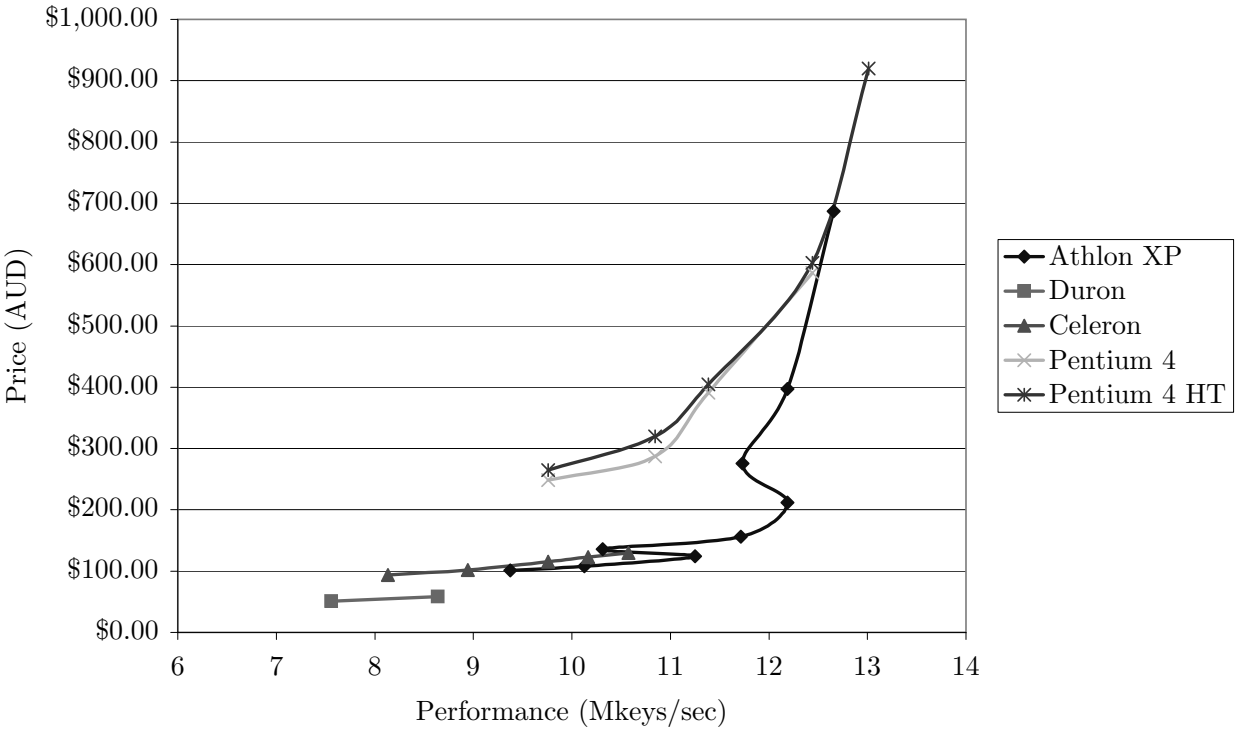

Figure 4.9: CPU price/performance by family for DES

would be useful when considering a very large-scale key search machine based on CPUs; these CPUs use much less power and generate far less heat. PIII-M, Centrino and G3 are particularly interesting due to their high benchmark results at comparatively low clock rates.

These comparisons ignore the cost of support hardware, which can be expected to be several times that of the CPU device itself in some case.

4.5.1 DES

Figure 4.9 shows the price and performance of each CPU family for the DES cipher. We can see that the CPUs within a family that achieve the highest search rates are disproportionately expensive. It is scarcely worth trying to achieve a search rate over 12Mkeys/sec with an Athlon XP or 11Mkeys/sec with a Pentium 4 because the price increases so steeply.

The Athlon XP curve contains a number of kinks; these occur because pricing increases with their performance rating. This performance rating is not in line with actual performance, however – Barton core Athlons have a higher performance rating than their clock speed (and measured performance) would suggest. We can see that the Duron line appears to fit reasonably well with the Athlon XP line. Celeron CPUs achieve higher performance than their price would otherwise suggest.

Examination of the data shows that the two Duron data points have a linear price/performance

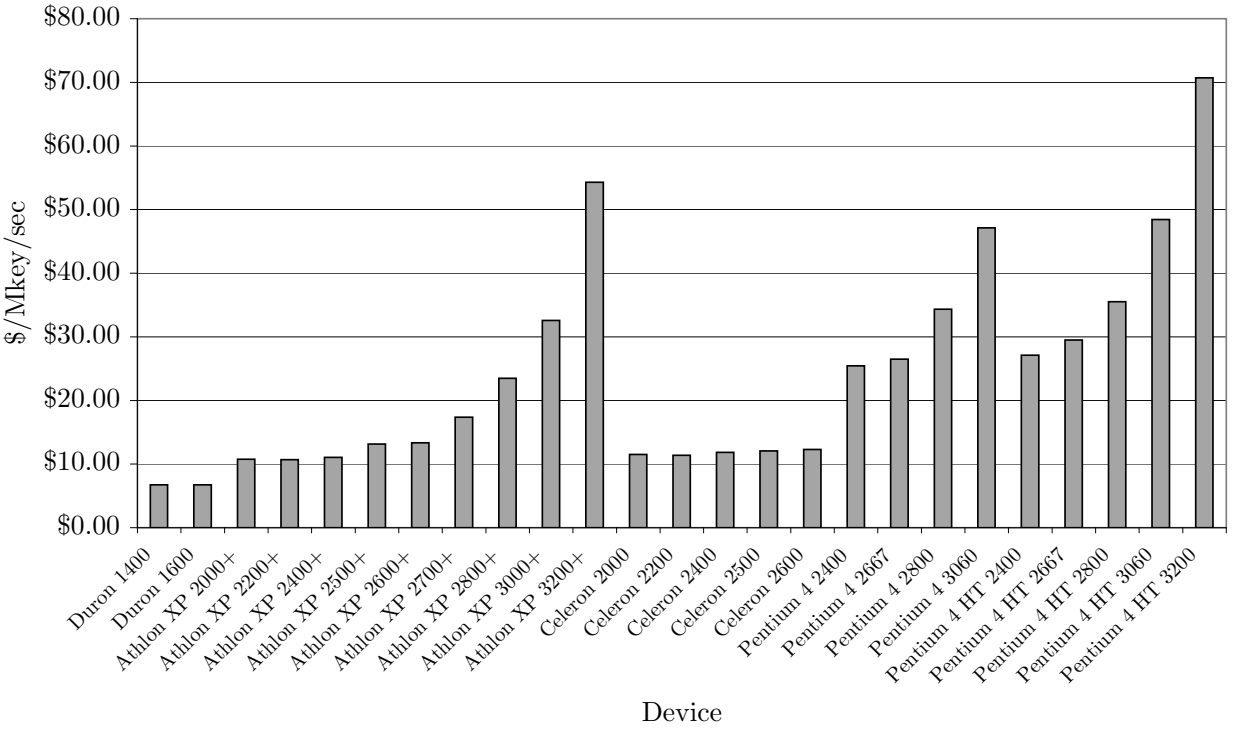

Figure 4.10: CPU price/performance by device for DES

relationship. In most practical systems, it would thus be best to choose the faster of the two in order to save on auxiliary costs (support hardware, space, etc.)

Figure 4.10 shows the price/performance ratio for each device under consideration. We can see that the slowest device in each family generally gives the best ratio. The Durons have exactly the same price/performance ratio. The Athlon XP 2200+ and Celeron 2200 provide a marginally better ratio than their neighbours. All Pentium 4 devices are quite expensive for the performance that they give. The exact ratio needed for a large-scale machine will depend on the price of the support hardware, but in general any Duron, Celeron or Athlon XP up to 2600+ will provide a good price/performance ratio.

4.5.2 RC5

Figure 4.11 shows the price/performance ratios of each CPU family for RC5. We can see that the Celeron and Pentium 4 families are far less competitive for RC5; the most expensive Pentium 4 HT device barely outperforms the cheapest Duron! The Duron and Athlon XP families remain similarly positioned relative to each other. The same kinks in the Athlon XP curve are apparent.

Figure 4.12 shows the price/performance ratios of each device for RC5. As with DES, the cheapest device in each family provides the best ratio (with the minor exceptions noted

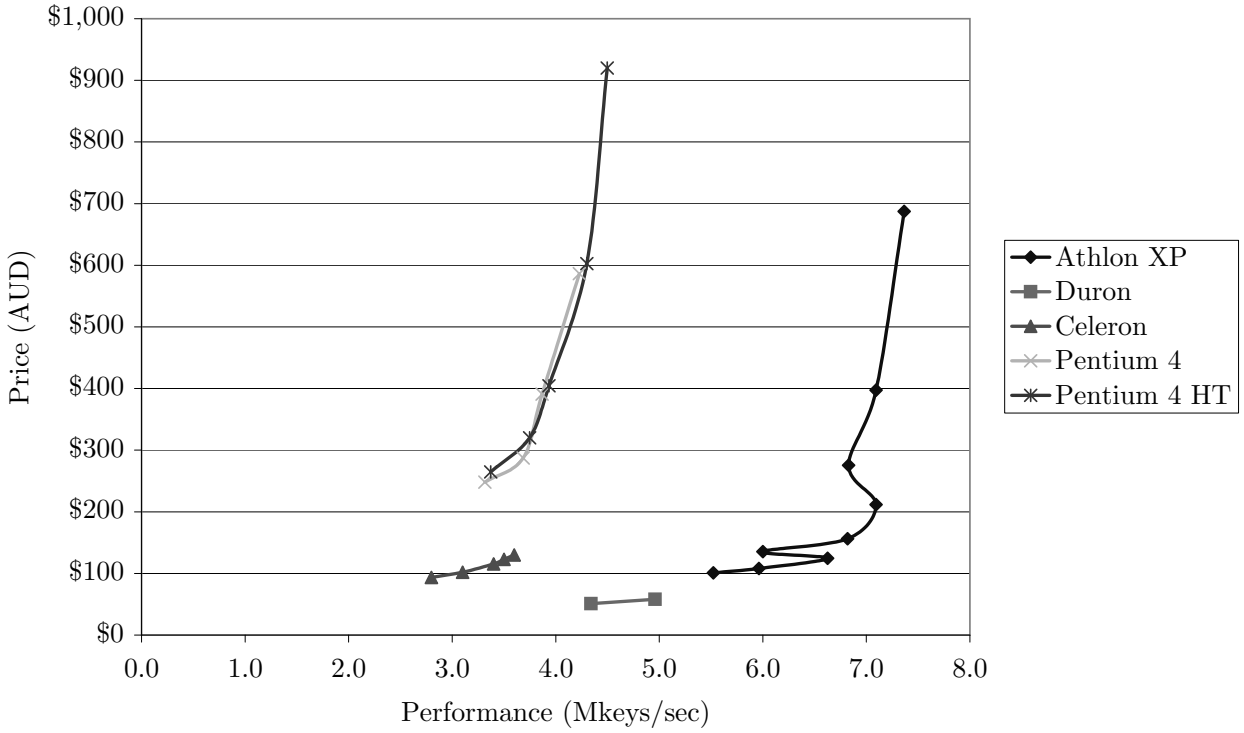

Figure 4.11: CPU price/performance by family for RC5

for DES). Unlike DES, however, the Celeron family is no longer as competitive. Key search machine designers would do best to select the Duron 1600 or a low-end Athlon XP. Both Pentium 4 varieties remain very expensive for their performance.

4.6 Technology comparison

4.6.1 CPUs and FPGAs

4.6.1.1 Ciphers

The pricing and performance data for CPUs and FPGAs is not directly comparable. CPU prices are given in AUD for quantities of 10; FPGA prices are given in USD for quantities of 24-99. The CPU performance is based on benchmark results, while the FPGA performance is based on synthesis estimates.

Nonetheless, we can scale the CPU pricing based on the current exchange rate, and scale the FPGA performance based on measured performance. At the time of writing, one Australian dollar is worth 0.700639 U.S. dollars. The predicted performance for the XCV1000E running DES was 894Mkeys/sec; achieved performance was 500Mkeys/sec. The DES CPU performance also needs to be scaled up by approximately 2.5 to account for the

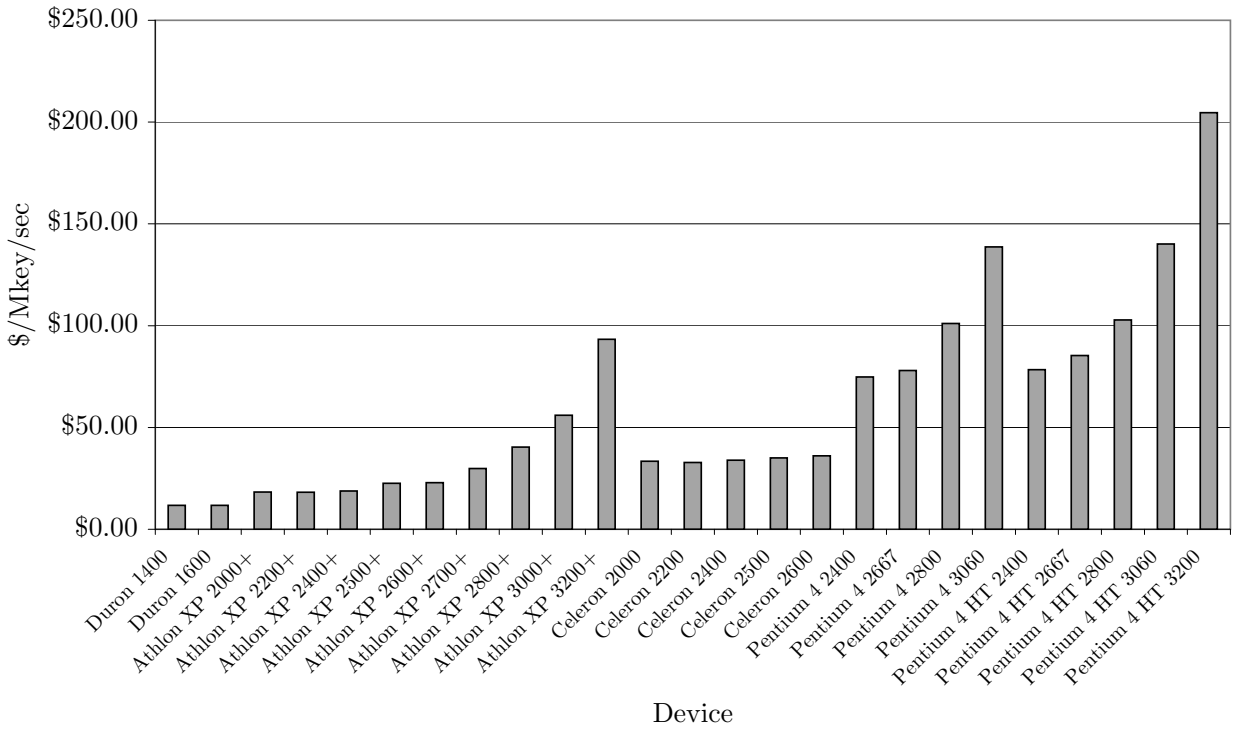

Figure 4.12: CPU price/performance by device for RC5

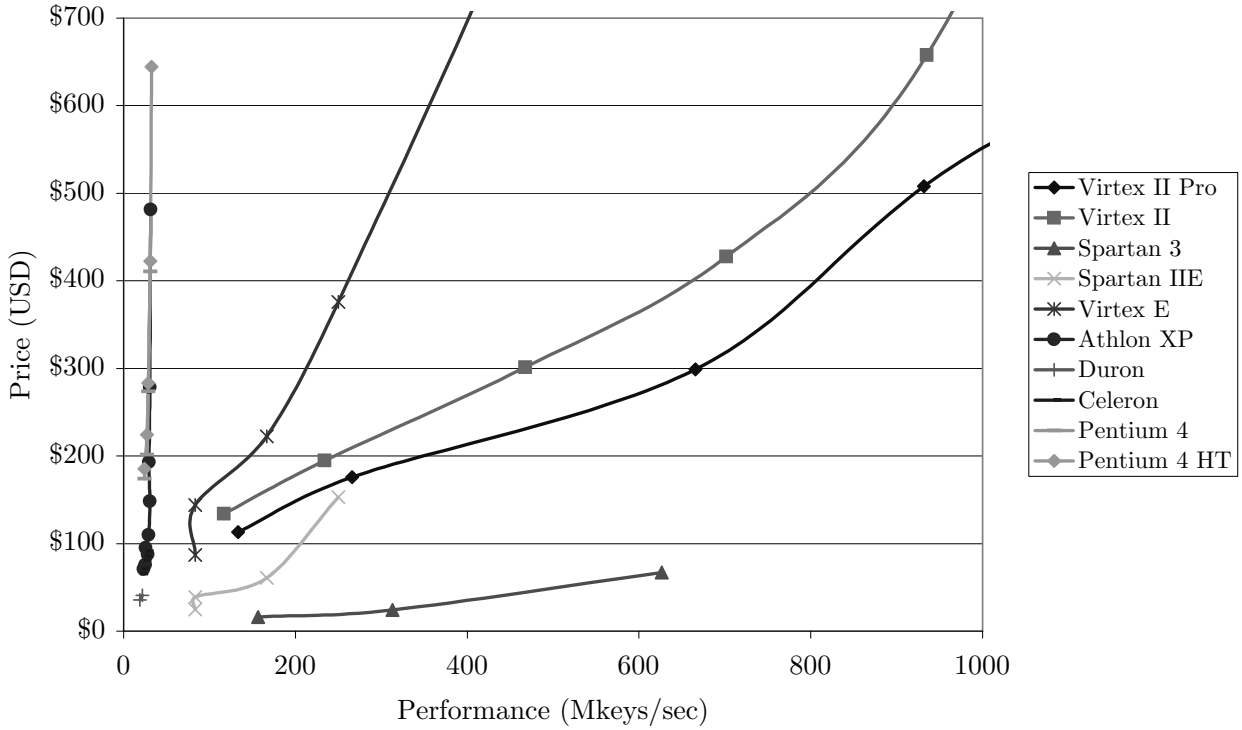

Figure 4.13: CPU and FPGA family comparison for DES

low speeds achieved by the SolNET client compared with the distributed.net client. The predicted FPGA performance for RC5 matched quite closely with the achieved performance (predictions were 1.0625 times faster.) Scaling with these figures ignores many factors but will suffice for this analysis.

Figure 4.13 shows the price/performance ratios for each CPU and FPGA family. The entire CPU range is compressed into the left-hand side of the graph; even at the high end, they do not come anywhere near the search rate of a low-end FPGA. It can be seen that searching DES on general purpose CPUs is very costly compared to searching with FPGAs.

Figure 4.14 shows the same comparison for the RC5 cipher with the less competitive FPGA families removed. CPUs now perform better than FPGAs at the same price. They still cannot match the performance offered by high-end FPGAs.

These two comparisons show that the choice of implementation technology can greatly affect the time and cost to perform an exhaustive key search. In a practical key search machine, the technology must be selected to match the cipher being attacked.

4.6.1.2 In general

Over time, FPGAs will most likely become more efficient for key search than CPUs. This is because CPU performance does not scale linearly with available silicon area; it is limited by

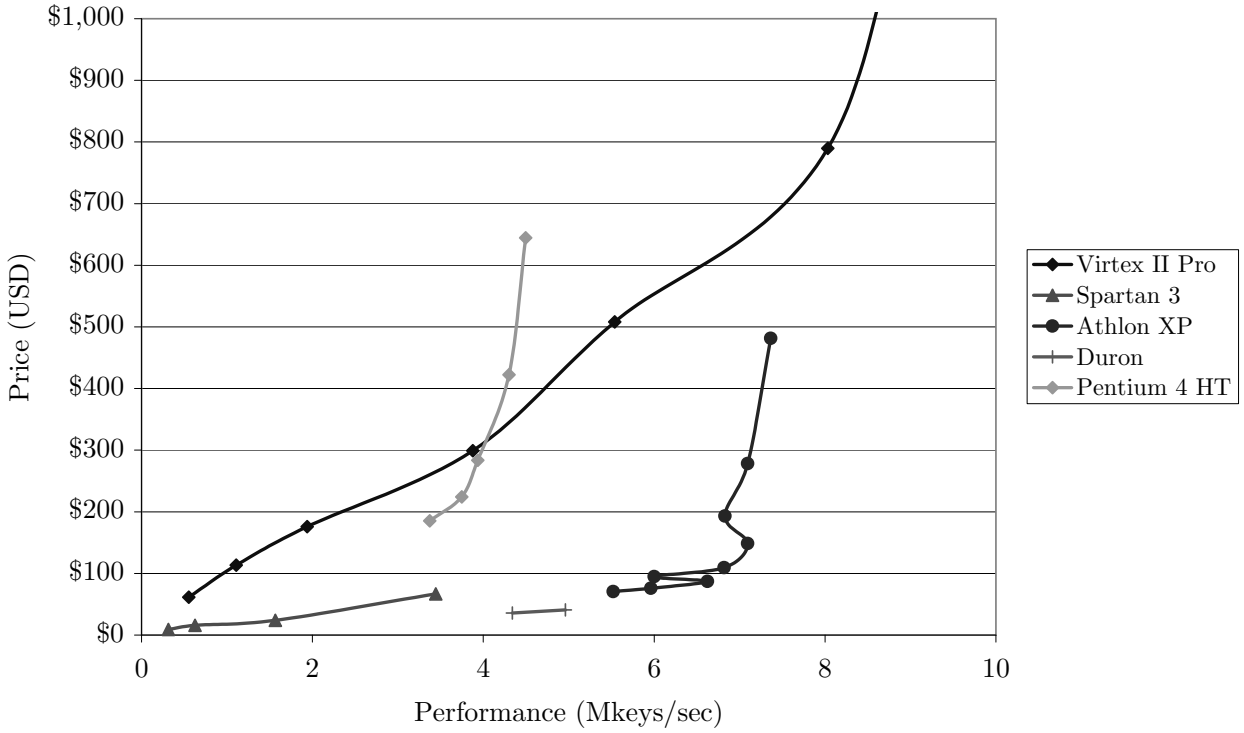

Figure 4.14: CPU and FPGA family comparison for RC5

bus speeds, interactions between instructions and limited parallelism. FPGA performance for key search will scale linearly; if twice as much silicon area is available, twice as many search units can be implemented. FPGAs will thus become more important in future cryptanalysis. Already, improved CPU performance is becoming dependent on increasing parallelism; SIMD techniques and HyperThreading are examples of this.

CPUs are, of course, far easier to obtain than FPGAs. Many organisations already have a large computing infrastructure that could be used to perform key searches. distributed.net and other software RC5 efforts have demonstrated the feasibility of this approach. FPGA hardware is very rare in comparison, especially in the quantities that would be needed to conduct key searches.

CPUs need a large amount of support hardware (heatsinks, RAM, chipsets, multi-voltage power supplies and so on) which drives up the cost of a CPU-based key search machine. All of this hardware is very cheap and available. Storage space for it becomes more of a concern. FPGAs have a clear advantage here; many FPGAs can be mounted on a card that will fit within a computer case. It may be possible to design a motherboard for commodity CPUs that has some of these advantages.

Ultimately, neither CPUs nor FPGAs are very efficient for conducting key searches compared with ASICs. CPUs are inefficient due to their support hardware and program-based

operation; FPGAs are inefficient because of their generic hardware structure. ASICs have custom hardware and a low unit price, but very high initial price.

4.6.2 Extrapolation for ASICs

In [48], Craig Ulmer reports that ASIC implementations can achieve three times the speed of an FPGA implementation, and ten times the density. This is useful as a general guide, but not in this analysis since the area required by FPGA dice is not easily obtained.

Instead, we can estimate ASIC costs based on the gate count required and infer the cost of a gate array device. During the Map phase of FPGA compilation, ISE reports the 'equivalent gate count' for an ASIC implementation of the design. This is based partly on the data contained within [49], and can be used to determine an approximate ASIC cost. The DES implementation on the XCV1000E uses 453,968 gates, and the RC5 implementation uses 1,353,397 gates. RC5's large gate count is due to the amount of RAM used, including the additional RAM blocks used to reduce routing delays on the FPGA implementation. Both of these figures include interface and controller logic. It would also be possible to find a tradeoff between die size with final cost.

According to [50], a Virtex E design should be implementable with a CMOS-10HD gate array. This is designed around a 250nm process, which seems reasonable for a 1:1 speed and density conversion; the Virtex E family uses a 180nm process. No measures on the physical size of a CMOS-10HD die were available, but [51] claims 15k gates/mm^2 for a CMOS-9HD die. Assuming that the number of gates per mm^2 scales linearly with feature size, we get approximately 30k gates/mm^2. Assuming a 50% gate utilisation ratio gives us approximately 900kgates for DES, or 30mm^2.

MOSIS [52] provides small-quantity ASIC fabrication. They also provide an online price list [53]. We select the TSMC 250nm process (CL025) as one that should be suitable; other 250nm processes are approximately the same price. This gives a fabrication cost of $44,200 for 40 parts. $2,500 more will be required for packaging [54]. This is a high average price per part, but not a great deal more than the price of the XCV1000E. No pricing data was available for larger quantities.

Without further pricing data, it is difficult to perform an intelligent cost comparison involving ASICs. We can, however, use the price of packaging as a bare minimum cost per device to determine a price point at which ASICs become a viable option. Figure 4.15 shows ASICs, CPUs and FPGAs compared at their best price points for the DES cipher. We can see that to assemble a machine equivalent in power to the EFF DES Cracker, FPGAs remain the most cost-effective choice. ASICs do not become price-effective until the machine performance reaches almost 400Gkeys/sec – over four times the performance of the

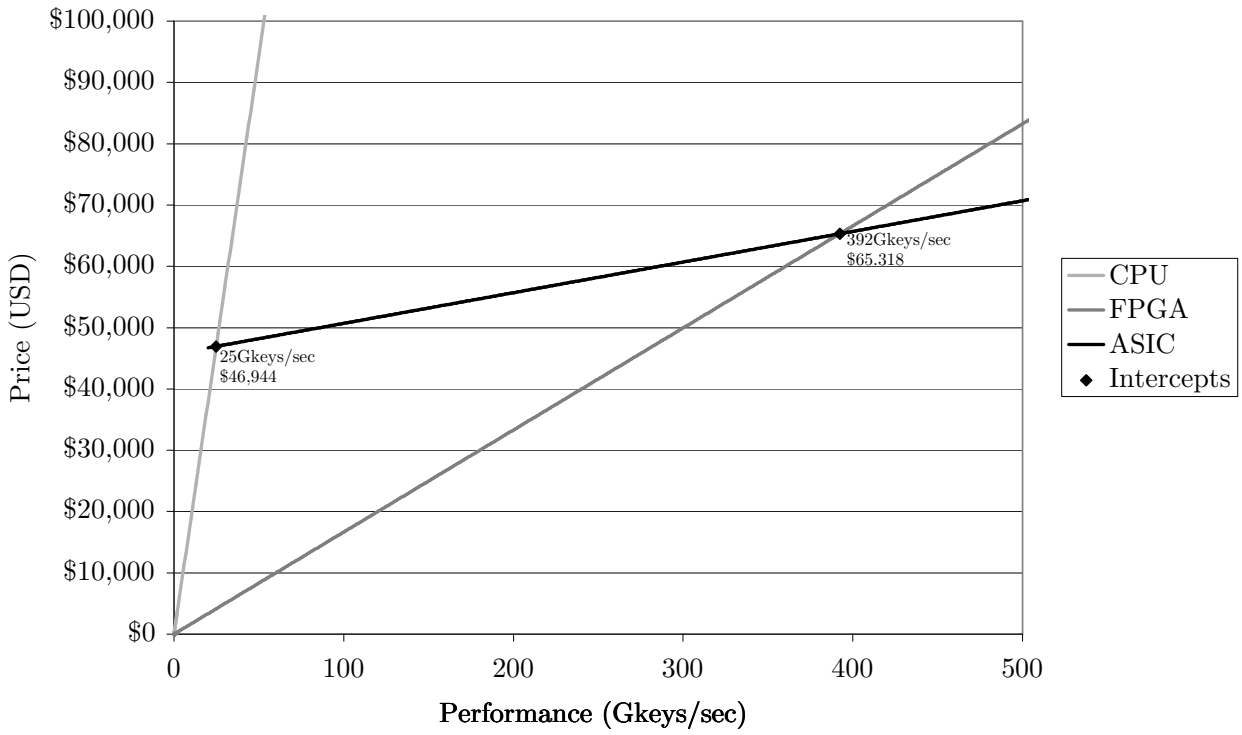

Figure 4.15: Comparison of CPUs, FPGAs and ASICs for DES

EFF machine.

Spartan 3 FPGAs were not included in this comparison due to their uncertain pricing and performance. In addition, their $/Mkeys/sec ratio is below that of the ASIC design given, meaning that the two curves would never converge (as they would if all fabrication options had been considered.) It will be interesting to update this analysis when Spartan 3 pricing stabilises.

4.7 Comparison with other DES FPGA results

Figure 4.16 shows known FPGA DES key search machines and the performance that was predicted by Blaze et al. in [2]. Extrapolating their estimates with Moore's Law gives an estimate of 1120Mkeys/sec for a $200 FPGA today. Performance estimates for the FPGAs priced around $200 are also shown.

The graph shows that no implementation has matched the performance predicted in 1996 for FPGA devices, regardless of the price of FPGA device used. The implementation presented in this thesis moves closer to predictions (as a percentage of expected performance) but still falls short. It also uses an FPGA that costs $938 today, well in excess of the $200 quote given. Other $200 FPGA devices are predicted to achieve similar performance.

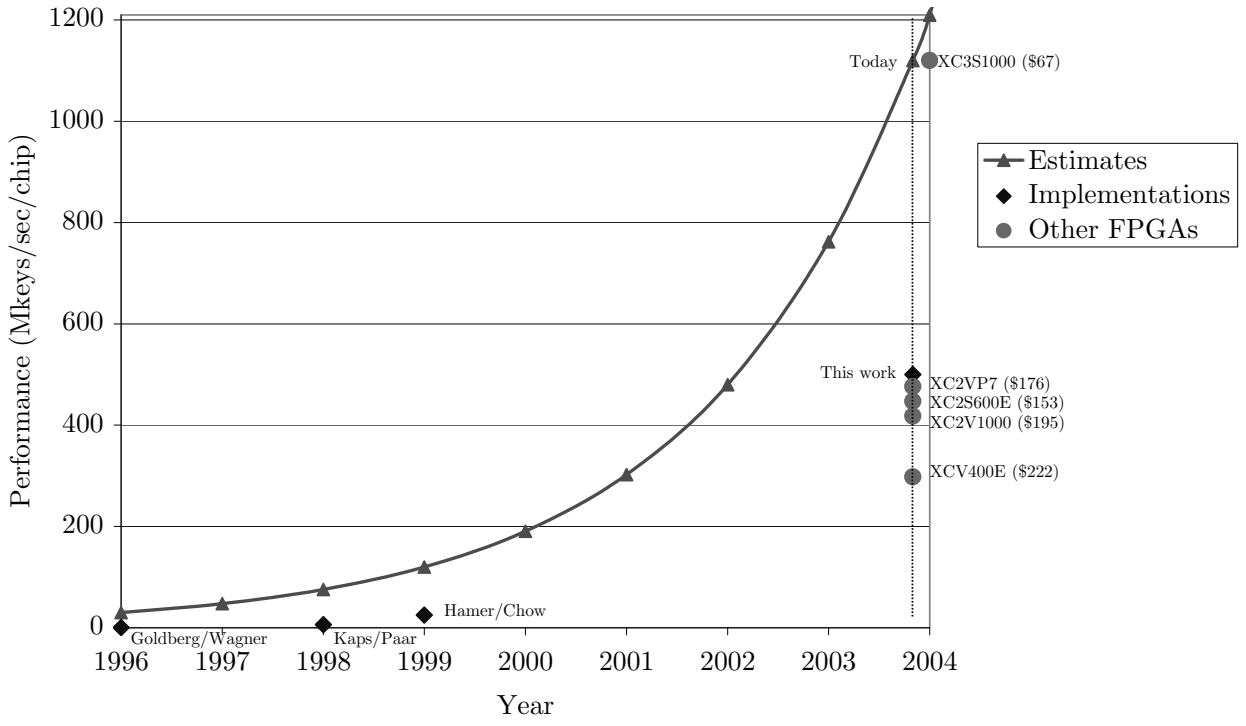

Figure 4.16: Previous FPGA DES key search machines and performance estimates

The XC3S1000 is interesting; it has a very high capacity for its price. It still falls short of the estimate, but not by much. Its predicted price is also well under $200. It will be interesting to see if this price remains accurate in Q1 2004. No larger Spartan 3 devices are shipping yet, so a device with a price closer to $200 could not be selected.

4.8 Large-scale key search machines

4.8.1 CPUs

CPUs are more suited to RC5 key search than FPGAs. A large-scale machine to complete the RSA RC5-72 challenge in one year might be considered. This requires an average search rate of almost 150Tkeys/sec to conduct a complete sweep of the key space (half a year on average to find the key). The most cost-effective CPU is the Duron 1600, achieving 5.0Mkeys/sec at a price of $58. To reach the target search rate, almost 30 million CPUs will be needed, costing over $1.7 billion. This is before considering extras such as RAM, motherboards, power, heat removal and storage space.

At present, distributed.net is achieving a key rate of approximately 120Gkeys/sec [55]. At the current rate, the RC5-72 challenge will probably be solved in 624 years.

A more feasible machine might attempt to match the performance of the EFF DES

Cracker, which achieved 92.6Gkeys/sec using a large number of gate array ASICs. The most cost-effective CPU is again the Duron 1600, achieving 21.5Mkeys/sec (distributed.net scale) for $58. Over 4300 CPUs would be needed at a cost of over $250,000. This is not excessively expensive, but again ignores support hardware and other extras.

4.8.2 FPGAs

FPGAs perform extremely well for DES key search. A machine matching the speed of the EFF DES Cracker could be constructed from XC2S200E devices ($25, 149Mkeys/sec). Spartan 3 devices were not considered due to their unstable pricing. 622 devices would be needed, at a total cost of $15,540. The EFF machine spent $130,000 on materials; it is not clear how much of this was spent on ASIC fabrication.

Alternatively, XC2VP20 devices could be used. They are slightly more expensive at a given search rate, but far fewer devices would be needed. This would reduce auxiliary costs significantly. To match the performance of the EFF DES Cracker, a mere 78 devices would be needed. Each device costs $299, giving a total component cost of $23,322. In contrast, the EFF machine used 1536 devices spanning many circuit boards and several physical cabinets.

At the top end of the FPGA spectrum, XC2VP100 devices could be used. These are the largest Xilinx devices that are currently shipping. Only 16 devices would be required. The total device cost would be $89,264, but the physical space consumed by the machine would be very small – less than that of a single board in the EFF DES Cracker.

FPGAs are generally more expensive than CPUs when performing RC5 key searches, and so will not be considered. Using the theoretical pipelined design may be profitable; a single (expensive) FPGA could search 100–200Mkeys/sec.

4.8.3 Accounting for hardware costs

The device cost of a large-scale key search machine is not the only factor affecting a machine's cost. All technologies require circuit boards, controllers, assembly, testing, power, storage and cooling considerations to be addressed.

4.8.3.1 ASICs and FPGAs

ASIC and FPGA implementations can use the estimates provided by Wiener [15]. A circuit board that can support 120 small package ICs is reported to cost $300. The devices used by Wiener are 18mm square. FG676 packages (as used on the XC2VP20) are 27mm square, fitting approximately 35 devices per board. PQ208 packages (as used on the XCS200E) are approximately the same size (28mm.) The microcontrollers used by Wiener are not needed

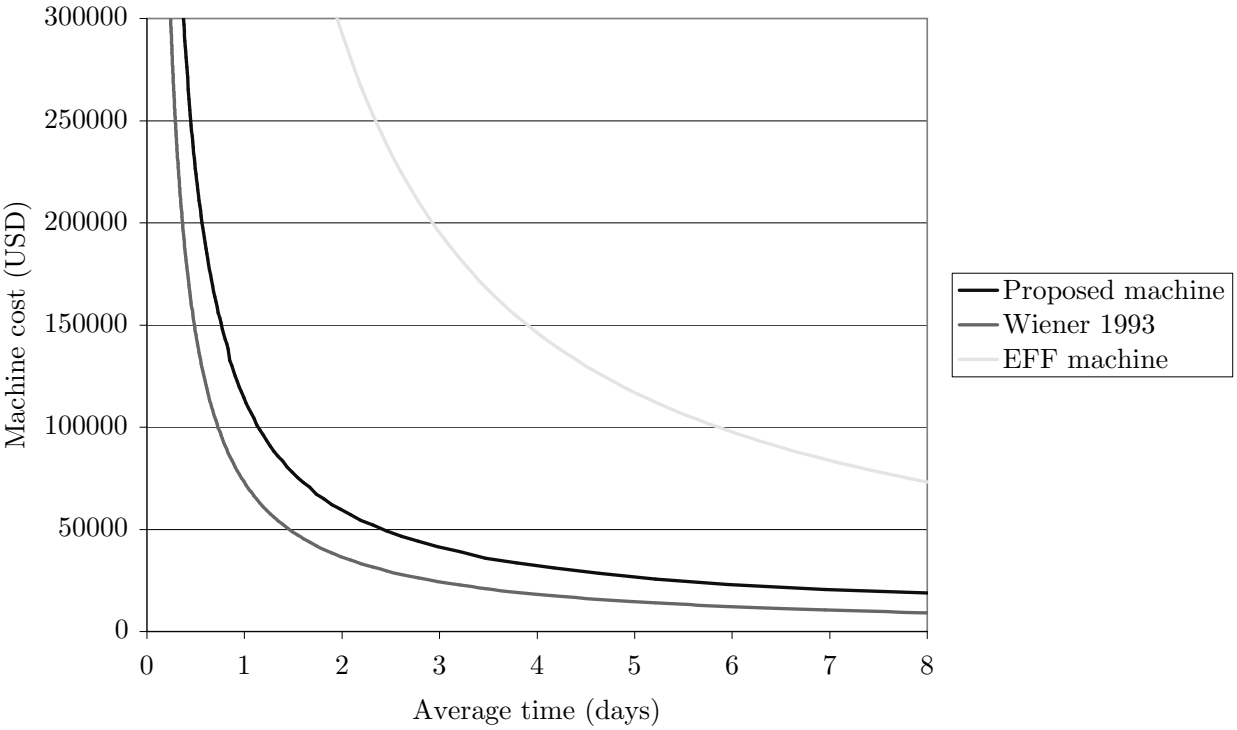

Figure 4.17: Cost of key search machines and their expected search time

since controllers can be integrated into the FPGAs. Assuming that only one FG676 or PQ208 can fit into the space occupied by four of Wiener's ASICs allows 35 devices per board.

From this we can see the value of high-density devices. One XC2VP20 has eight times the performance of an XCS200E. A machine capable of 100Gkeys/sec would be take just over four days on average to find a key. Taking into account circuit board costs, a machine using XCS200E devices would span 671 devices, 20 boards and cost $22,641. A similar machine using XC2VP20 devices would span 84 devices, three boards and cost $26,016. Factoring in controllers, power supplies and mechanical concerns according to Wiener's figures gives a total of $33,741 for the XCS200E machine and $33,816 for the XC2VP20 machine. Power consumption, heat generation and storage space for the XC2VP20 machine would be significantly lower at only a very small increase in total price.

Using the preliminary pricing for XC3S1000 devices gives 89 devices, three boards and $13,663. Most of the cost in this machine is devoted to auxiliary hardware, meaning that higher speed machines would cost less in relation to the key rate achieved.

Figure 4.17 infers the cost to find a DES key in a given amount of time. Wiener's estimates, the EFF DES Cracker and the estimates proposed by this work for XC2VP20

devices are shown. We can see that to achieve very short search times a lot of money must be spent, and vice-versa.

4.8.3.2 CPUs

Commodity computer hardware pricing is needed to infer the remainder of the hardware costs. Assuming that computers can boot from a network, each CPU will need a heatsink, motherboard, case, power supply, network adapter and a small amount of RAM. Using an all-in-one motherboard and low quality case reduces costs significantly. Each CPU will need approximately AUD$175 in support hardware.

4.9 Key lengths

It has been shown time and time again that using a long key is the best way to protect a cipher from an exhaustive key search attack. Assuming a cipher with a 90 bit key length (as recommended in [2]) that can be attacked at the same speed as DES, 132 billion XC2S200E devices would be needed to cover the key space in a year, at a price of $3.3 trillion. Even using the best available Spartan 3 device (XC3S400) will cost $850 billion and need over 35 billion devices. Of course, attacks become even more infeasible if the key length is increased only slightly.

4.9.1 Capabilities

Well-resourced entities can feasibly attack ciphers that use long key lengths. Assuming the performance of the XC2VP20 machine is maintained regardless of key length, we can see the cost to obtain a key within a year in Table 4.3. If an attacker is prepared to wait for a year, 56 bit ciphers like DES are trivial to defeat using current technology. Each additional bit added to the key doubles the cost to break it within one year.

4.9.2 Minimum key lengths

The minimum key length required for a system depends on who the potential attackers will be. Assuming that we want messages encrypted today to be completely undecipherable to all known attackers, a 92 bit minimum key length seems to be appropriate.

Future data security must also be considered. Moore's Law is currently the de facto method of predicting future computing capabilities. Over an 18 month period, Moore's Law states that transistors per IC (or computing power) will double. If we apply this to a 20 year period, messages must be encrypted with 14 bits of additional key length to remain secure

Key length	Cost	Potential attacker
56	$305	Bored teenager
60	$4,900	Employed adult
64	$78,000	Business department
68	$1.25 million	Large business
72	$20 million	Small government
76	$320 million	Large government entity
80	$5.1 billion	Significant inter-government collaboration
92	$21 trillion	Infeasible?

Table 4.3: Cost to obtain a key in one year and potential attackers

against all known potential attackers. 106 bits appears to be a suitable minimum key length to keep data secure for the next 20 years.

Data that needs to be kept secure over a longer period of time (such as census data) will need an even longer key. To keep data secure for the next hundred years, a 159 bit key seems appropriate.

All of these estimates ignore future computing technologies that may become available or new cryptanalytic techniques which may decrease the strength of a given cipher. Predicting suitable key lengths can be likened to telling the future. Given the low additional processing cost, using a key length of 192 or 256 bits should protect data from all known potential attackers in the foreseeable future.

4.9.3 Alternatives

When attacking a well-designed system that uses cryptography, it is rarely profitable to attack the ciphers themselves. Normally there are weaker areas of the system, such as software bugs, inadequate security policies, weak passwords and the people involved in the system. It will almost certainly be easier to exploit one of these areas (particularly people) to complete an attack against a system.

4.10 Regulatory issues

One of the strongest reasons that this research is valuable is in the context of legal restrictions on the use and export of strong cryptography. With it we can evaluate different ciphers in the face of key length restrictions, as well as the availability of the technology required to perform an exhaustive key search attack.

4.10.1 Cryptography controls

Unless otherwise referenced, information in this section was assimilated from [56], [57] and [58]. They should be consulted for more details.

Local regulations on cryptography have changed significantly in recent years. Australia is a party to the Wassenaar Agreement, which restricts the export of "dual-use goods", including encryption. It is vague in parts (particularly as to what constitutes an "export"), but sufficiently restrictive to raise concerns. Australia's regulations are more restrictive in that any export requires approval from the Defence Signals Directorate (DSD), which deals with Australia's signals intelligence and information security [59]. The *Defence and Strategic Goods List [60]* describes goods which may be subject to export controls, including encryption. Many different products are covered by the legislation, including nuclear, biological, optical, semiconductor and other technology goods. It is frequently updated.

Obtaining export approval usually involves submitting an export application [61] with the DSD. To date no applications have been rejected, although some companies have been informed that their applications will be rejected without having applied. An early assessment of cryptographic goods can be performed to determine if export approval will be required for that good [62].

There are no restrictions on the of cryptography within Australia or the importation of cryptography into Australia.

The United States has comparatively tight controls on cryptographic exports. Currently, symmetric cryptography using up to 56 bit keys is able to be exported once it has undergone a one-time review. Export of any cryptography is not permitted to seven "terrorist countries" (also known as "Tier 4" [63].) As can be seen in the results from this thesis, 56 bit symmetric cryptography is not very resistant to brute force attacks. Someone operating under these constraints would be advised to select a cipher that is relatively slow and expensive to attack, such as RC5.

The legal export status of this thesis and its accompanying CD can be questioned. The thesis itself is probably safe to export without approval, since it does not contain any cryptographic algorithms. The CD can almost certainly not be exported without approval, since it contains cryptographic source code.

4.10.2 Computing controls

Regulatory issues exist for exports of high performance computers to certain countries. This was most visible when exports of the Playstation II gaming console to China were denied [64] for fears that they may enhance China's military capability. The regulations have recently been updated to increase the allowed performance of exported devices [65]. Computer

exports are controlled for "Tier 3" countries, which generally includes any countries that are not allied with the United States. Exemptions can be obtained to bypass these controls.

It is not clear whether FPGA devices are subject to export controls, but it would almost certainly be easy to force designs to fall under various performance classifications.

Chapter 5

Conclusion

From the analyses presented in Chapter 4, we can see that:

- Key length, available resources and cipher design are the main three factors that influence the time taken to conduct an exhaustive key search attack.
- Different implementation technologies favour different ciphers. DES key searches are best performed with FPGAs, while RC5 key searches are best performed with CPUs.
- The resource usage of a pipelined FPGA cipher implementation is dependent on the frequency of register access, state size, number of rounds and complexity of the round function.
- Frequency of register access is one of the biggest factors affecting resource usage for pipelined FPGA cipher implementations.
- If sufficient FPGA resources are available, pipelined cipher implementations will perform far better than iterative cipher implementations.
- Based on preliminary pricing, Spartan 3 FPGAs (particularly the XC3S400) have the best price/performance ratio.
- Based on stable pricing, Spartan IIE FPGAs (particularly the XC2S200E) have the best price/performance ratio, followed closely by the XC2VP20 (which has a much higher density).
- Duron and low-end Athlon XP CPUs provide the best price/performance ratio.
- The performance estimates in [2] are most likely to be optimistic. This view is shared by Golberg and Wagner [16].

- The cost of conducting a ciphertext-only attack with FPGAs depends on the cipher. The additional resource cost is quite small for DES, but significant for RC5. Ciphertext-only attacks favour large, fast search units over small slow search units.
- A machine similar to the EFF DES cracker [10] could be built from FPGAs for approximately $34,000, a fraction of the price of the original machine.
- CPUs are more cost-effective for RC5 key searches than FPGAs, although it remains to be seen whether this remains true for a pipelined RC5 implementation.
- Cryptography that is restricted to a 56 bit key length by export controls provides little protection against a well-funded or patient adversary.
- FPGAs will play a greater part in cryptanalysis in the future.

From these conclusions, we can see that in the right situations FPGAs are very useful cryptanalytic tools. Their low price and high performance allows key search attacks to be conducted at very low cost. If physical space devices is a concern, they can achieve much higher search rates per device than CPUs, even for ciphers that are designed for CPUs.

The EFF DES cracker can be reproduced now using FPGAs at a cost of about $34,000. At a price this low, DES should not be used for anything remotely secure. Government concessions to allow the export of 56 bit cryptography completely destroy the purpose of using cryptography.

FPGAs will play an increasing role in future cryptanalysis as the gap between CPU and FPGA performance for a given price widens.

5.1 Future work

Possible extensions to this work include:

- Update the price/performance analyses as time progresses. This would allow the security of ciphers to be continually tracked and give a general idea of the rate of improvement in FPGA and CPU technology.
- Analyse more ciphers and determine their resistance to exhaustive key search using various technologies.
- Examine DSPs and CPLDs as possible low-priced technology alternatives.
- Improve the DES benchmark software. The programs used for benchmarks were not designed with modern CPUs in mind, and may be able to achieve very high performance

by taking advantage of available features. In particular, SIMD architectures such as Altivec and SSE2 may prove useful.

- Implement RC5 as a long pipeline. Estimates show that this may result in very high search rates. High capacity FPGA devices would be needed to attempt this.
- Examine different FPGA families. No Altera devices were considered for this thesis. Actel produces a gate array family called the Axcelerator [66] which is one-time programmable and is reported to have very low routing overheads and a low price. Spartan 3 devices should also be re-examined once better pricing data becomes available.
- Improve the accuracy of the price/performance estimates for ASIC devices. Different fabrication processes may provide better price/performance ratios.
- Extend the FPGA resource estimation techniques to include timing data. With careful analysis, it should be possible to approximate overall performance given a cipher algorithm.
- Consider heat generation and power usage for FPGAs. One of the problems encountered with the EFF machine was the high power and cooling requirement for the machine. FPGA devices are reported to be inefficient in this regard, which may prove a stumbling point for large-scale key search machines.

Appendix A

Key search engine 1 interface

A.1 Description

The interface allows the following operations to be performed:

- retrieve the status of the board
- retrieve potential keys
- set the ciphertext, plaintext and IV to be used by all search units
- access and detect all search units controlled by the machine
- obtain the status of a single search unit
- set or retrieve the next key that will be processed by a search unit

A.2 Registers

The controller provides a number of registers which allow the computer to access the machine's resources.

Address	Name	Description	Read/write
0	STATUS	Status register	Read only
1	NEXTKEY	Retrieves the next potential key from the buffer	Read only
2	CTEXT	Sets the known ciphertext	Write only
3	PTEXT	Sets the known plaintext	Write only
4	SUSEL	Select a search unit	Write only
5	SUKEY	Set or retrieve the current value of the key generator	Read/write
6	IV	Sets the initialisation vector	Write only

Table A.1: Initial key search machine registers

Only the least significant 3 bits are decoded. All transfers are 64 bits wide. This makes supporting key lengths greater than 64 bits difficult.

When reading the STATUS register, the least significant word contains the following bits:

Bit	Name	Description
0–3	VERSION	Described below
4	BUFFER_FULL	Set when the key buffer is full
5	DVALID	Set when the machine is ready for a new command
6	SU_PRESENT	Set when the currently selected search unit exists
7	SU_RUNNING	Set when the currently selected search unit is running
8	BUFFER_EMPTY	Set when the key buffer is empty

Table A.2: Initial key search machine STATUS register

VERSION specifies the version of the communication protocol. For this iteration of the design, the version is "0001". It is also used to detect whether the board is programmed and operating properly. As such, it should never be "0000". This catches the case where the FPGA has not been correctly programmed.

DVALID is set when the machine is ready for a new command, and cleared when a command is currently executing. Results from a write command should not be read until DVALID is set.

When written to, the SUSEL register selects a search unit. Any subsequent commands that operate on a specific search unit operate on the search unit specified in SUSEL. Writing to SUSEL also updates the value of the SUKEY register and the SU_PRESENT and SU_RUNNING bits in the STATUS register. The SUSEL register must be repeatedly written to in order to keep this data up to date.

SU_RUNNING is set when the last selected search unit is running, and cleared when the search unit is halted. A search unit might be halted if it has found a key and is waiting to have the key read, or if no initial key has been set.

A.3 Operation

1. Software checks presence and version of board by reading STATUS register
2. If VERSION is 0, program complains that FPGA has not been programmed
3. If VERSION is not 1, program complains that software version does not match or FPGA is incorrectly programmed
4. For each address where a search unit is believed to exist

   (a) Software writes the address to SUSEL

   (b) Software polls STATUS until DVALID goes high

   (c) If SU_PRESENT is 1, the search unit exists and can be used; if 0, search unit does not exist

5. Software writes CTEXT, PTEXT and IV registers

6. For each search unit:

   (a) Software waits for DVALID flag to go high

   (b) Software selects a search unit (sets SUSEL register)

   (c) Software writes initial key into search unit (writes SUKEY register)

7. Until correct key is located:

   (a) Software polls STATUS register to determine if any potential keys have been located

   (b) If there is a pending key, read it out of the buffer

The key that is written into the key buffer is always the key that was in the key generator at the time the search unit was halted, not the key that caused the search unit to halt. The software must be aware of the number of clock cycles required to process a single key, and subtract that value from the value stored in the key buffer. This value is algorithm dependent.

Appendix B

Key search machine 2 interface

B.1 Registers

The registers available to the programmer are:

Address	Name	Description	Read/write
0	BUFFER	See text	Read/write
1	CTEXT	Sets the ciphertext to use	Write only
2	PTEXT	Sets the plaintext to use	Write only
3	IV	Sets the initialisation vector to use	Write only

Table B.1: Revised key search machine registers

The BUFFER register has the following format:

VERSION and DVALID function identically to the original key search machine (see Chapter A.) In this version of the machine the BUFFER register indirectly controls the search bus. A write is performed by setting RW to 1 and specifying the data in the DATA register. A read is performed by writing a word with RW set to 0 and then polling the

Bits	Name	Description
0–3	VERSION	Protocol version (2)
4	DVALID	Set when the machine is ready for a new command
5	unused	
6	RW	Specifies whether this command describes a read or write operation
7	unused	
8–15	ADDR	Specifies the target address for the write or read
16–63	DATA	See text

Table B.2: Revised key search machine BUFFER register

Bits	Name	Description
0	KEYVALID	Set when the search unit has a key block to search through
1	RUNNING	Set when the search unit is searching its key block
2–7	unused	
8–47	KEY	The least significant 32 bits of the key value

Table B.3: Revised key search machine search unit read format

BUFFER register until DVALID goes high. The data will be contained in the space allocated to the DATA field.

A read or write through the BUFFER register always sets or retrieves the key in use by a search unit. The exact interpretation of the DATA field depends on the key generator in use. The intended purpose is for DATA to be interpreted as a block number during a write, and treated as the key number (least significant 32 bits) during a read.

When reading through the BUFFER register, 32 bits are used by the key value. The other 8 bits are used by the search unit to report status information:

B.2 Operation

1. Software checks presence and version of board by reading BUFFER register
2. If VERSION is 0, program complains that FPGA has not been programmed
3. If VERSION is not 2, program complains that software version does not match or FPGA is incorrectly programmed
4. For each address where a search unit is believed to exist
   (a) Software performs a read-through-BUFFER operation on the appropriate address
   (b) If the key returned is 1, a search unit exists at that address
5. Software writes CTEXT, PTEXT and IV registers
6. For each search unit:
   (a) Software writes initial key into search unit with a write-through-BUFFER operation
7. Until correct key is located:
   (a) Software polls the RUNNING bit of each known search unit in turn.

(b) If RUNNING on a search unit is 0:

i. Record the key value as a potential key

ii. Write the block number to the search unit so that it continues searching from the same point

The key that is read from the key buffer is the value that was in the key generator at the time the search unit was halted, not the key that caused the search unit to halt. The software must be aware of the number of clock cycles required to process a single key and subtract that value from the retrieved value. This value is algorithm dependent.

Appendix C

CPU benchmark results

Processor	Clock rate (MHz)	Core	Speed (Mkeys/sec)
Pentium IV	2533	SolNET (BrydDES)	10.3
Athlon 2500+ (Barton)	1833	d.net (Byte Bryd)	10.2
Athlon XP 1900+	1600	SolNET (BrydDES)	9.0
Pentium IV	1800	SolNET (BrydDES)	7.5
Pentium IV-M	1700	SolNET (BrydDES)	7.5
Duron	1000	SolNET (BrydDES)	5.4
Pentium III M	1000	SolNET (BrydDES)	4.3
Pentium MMX	200	d.net (MMX bitslice)	2.9
Pentium II	233	d.net (MMX bitslice)	2.6
Celeron-A	450	SolNET (BrydDES)	2.0
Pentium II	233	SolNET (BrydDES)	1.1
Pentium MMX	200	SolNET (BrydDES)	1.0

Table C.1: DES software benchmark results

Processor	Clock rate (MHz)	Core	Speed (Mkeys/sec)
Athlon XP 2500+ (Barton)	1833	SS 2-pipe	6.0
Athlon XP 1900+	1600	SS 2-pipe	5.3
Pentium IV HT	3060	DG 3-pipe	4.3
Pentium IV	2533	DG 3-pipe	3.5
Duron	1000	SS 2-pipe	3.1
Pentium IV-M	1700	DG 3-pipe	2.4
PowerPC 740/750 G3	900	MH 1-pipe	2.3
Pentium III-M	1000	SES 2-pipe	2.1
Pentium III	533	SES 2-pipe	1.1
Celeron-A	450	SES 2-pipe	0.9
Pentium II	233	SES 2-pipe	0.5

Table C.2: RC5 software benchmark results

Appendix D

FPGA price/performance tables

The following resource and speed figures in D.1 were used to determine the final price/performance table.

FPGA pricing is specified in USD and was obtained from Avnet [45] on October 15th, 2003. In all cases, the cheapest package available was used; this was usually also the smallest package.

Spartan 3 devices only started shipping recently, and pricing is still highly unstable. No price could be obtained for the XC3S200 device.

RC4 performance figures use the same relative performance ratios as RC5; both are RAM-based cipher implementations. Resource figures were inferred from [21].

		RC5			DES		
Family	Speed	MHz	SU slices	SU RAM	MHz	SU slices	SU RAM
Virtex II Pro	-5	138	666	1	238	1774	0
Virtex II	-4	120	663	1	209	1774	0
Spartan 3	-4	156	657	1	280	1774	0
Spartan IIE	-6	103	700	2	149	1806	0
Virtex E	-6	103	700	2	149	1806	0

Table D.1: Relative FPGA family performance

				DES		RC5		RC4	
FPGA	Speed	Package	Price	Mk/s	$/Mk/s	Mk/s	$/Mk/s	Mk/s	$/Mk/s
XC2VP2	-5	FG256C	$62	0	–	0.59	$104.59	1.02	$60.63
XC2VP4	-5	FG256C	$113	238	$0.48	1.18	$96.26	2.37	$47.83
XC2VP7	-5	FG456C	$176	476	$0.37	2.06	$85.45	3.72	$47.28
XC2VP20	-5	FG676C	$299	1190	$0.25	4.12	$72.58	7.44	$40.16
XC2VP30	-5	FG676C	$508	1666	$0.31	5.88	$86.36	11.51	$44.17
XC2VP40	-5	FG676C	$790	2380	$0.33	8.53	$92.56	16.24	$48.63
XC2VP50	-5	FF1152C	$1477	3094	$0.48	10.30	$143.45	19.63	$75.27
XC2VP70	-5	FF1517C	$2256	4284	$0.53	14.71	$153.35	27.75	$81.31
XC2VP100	-5	FF1696C	$5579	5712	$0.98	19.42	$287.29	37.56	$148.54
XC2V40	-4	CS144C	$22	0	–	0.00	–	0.29	$76.04
XC2V80	-4	CS144C	$28	0	–	0.00	–	0.59	$48.35
XC2V250	-4	FG256C	$79	0	–	0.51	$155.09	1.76	$45.16
XC2V500	-4	FG256C	$134	209	$0.64	1.02	$131.12	2.34	$57.27
XC2V1000	-4	FG256C	$195	418	$0.47	1.79	$108.71	2.93	$66.47
XC2V1500	-4	FG676C	$301	836	$0.36	2.81	$107.09	3.52	$85.74
XC2V2000	-4	FG676C	$428	1254	$0.34	4.09	$104.52	4.10	$104.34
XC2V3000	-4	FG676C	$658	1672	$0.39	5.37	$122.42	7.03	$93.57
XC2V4000	-4	FF1152C	$1552	2508	$0.62	8.70	$178.42	8.79	$176.62
XC2V6000	-4	FF1152C	$2936	3971	$0.74	13.05	$224.99	10.55	$278.40
XC2V8000	-4	FF1152C	$7446	5434	$1.37	17.91	$415.73	12.30	$605.21
XC3S50	-4	VQ100	$9	0	–	0.33	$27.06	0.38	$23.45
XC3S200	-4	VQ100	$16	280	$0.06	0.67	$24.05	1.15	$13.89
XC3S400	-4	TQ144	$24	560	$0.04	1.66	$14.43	1.54	$15.63
XC3S1000	-4	FT256	$67	1120	$0.06	3.66	$18.31	2.30	$29.09
XC2S50E	-6	TQ144C	$12	0	–	0.22	$56.35	0.51	$24.50
XC2S100E	-6	TQ144C	$15	0	–	0.22	$69.12	0.63	$24.05
XC2S150E	-6	PQ208C	$21	0	–	0.44	$46.96	0.76	$27.23
XC2S200E	-6	PQ208C	$25	149	$0.17	0.66	$37.65	0.88	$28.07
XC2S300E	-6	PQ208C	$39	149	$0.26	0.88	$44.45	1.01	$38.66
XC2S400E	-6	FT256C	$61	298	$0.20	1.32	$46.29	2.53	$24.15
XC2S600E	-6	FG456C	$153	447	$0.34	1.98	$77.36	4.55	$33.64
XCV50E	-6	CS144C	$33	0	–	0.22	$150.76	1.01	$32.78
XCV100E	-6	CS144C	$49	0	–	0.22	$224.62	1.26	$39.07
XCV200E	-6	CS144C	$87	149	$0.58	0.66	$131.98	1.77	$49.19
XCV300E	-6	PQ240C	$144	149	$0.97	0.88	$164.04	2.02	$71.33
XCV400E	-6	PQ240C	$222	298	$0.75	1.32	$168.63	2.53	$87.99
XCV600E	-6	HQ240C	$376	447	$0.84	1.98	$190.23	4.55	$82.72
XCV1000E	-6	HQ240C	$938	894	$1.05	3.73	$251.32	6.06	$154.82
XCV1600E	-6	BG560C	$1522	1192	$1.28	4.83	$315.10	9.09	$167.46
XCV2000E	-6	BG560C	$2142	1490	$1.44	5.93	$361.19	10.10	$212.03
XCV2600E	-6	FG1156C	$4620	2086	$2.21	7.91	$584.35	11.62	$397.72
XCV3200E	-6	CG1156CES	$6155	2533	$2.43	10.10	$609.21	13.13	$468.69

Table D.2: FPGA price/performance

Appendix E

CPU price/performance tables

				RC5		DES	
Family	Rating	MHz	Price	Mk/s	$/Mk/s	Mk/s	$/Mk/s
Athlon XP	1900+	1600		**5.3**		**9.0**	
Athlon XP	2000+	1667	$101	5.5	$18.27	9.4	$10.76
Athlon XP	2200+	1800	$108	6.0	$18.14	10.1	$10.68
Athlon XP	2400+	2000	$125	6.6	$18.80	11.3	$11.07
Athlon XP (Barton)	2500+	1833	$135	**6.0**	$22.58	10.3	$13.14
Athlon XP (Barton)	2600+	2083	$156	6.8	$22.93	11.7	$13.34
Athlon XP (Barton)	2700+	2167	$212	7.1	$29.86	12.2	$17.38
Athlon XP (Barton)	2800+	2086	$275	6.8	$40.34	11.7	$23.48
Athlon XP (Barton)	3000+	2167	$397	7.1	$56.01	12.2	$32.59
Athlon XP (Barton)	3200+	2250	$687	7.4	$93.32	12.7	$54.30
Duron		1000		**3.1**		**5.4**	
Duron		1400	$51	4.3	$11.73	7.6	$6.73
Duron		1600	$58	5.0	$11.73	8.6	$6.73
Celeron		2000	$94	*2.8*	$33.44	8.1	$11.51
Celeron		2200	$102	*3.1*	$32.85	8.9	$11.38
Celeron		2400	$115	3.4	$33.96	9.8	$11.83
Celeron		2500	$123	3.5	$35.07	10.2	$12.07
Celeron		2600	$130	*3.6*	$36.11	10.6	$12.30
Pentium 4		1800				**7.5**	
Pentium 4		2400	$248	3.3	$74.84	9.8	$25.43
Pentium 4		2533		**3.5**		**10.3**	
Pentium 4		2667	$287	3.7	$77.95	10.8	$26.49
Pentium 4		2800	$391	3.9	$101.04	11.4	$34.33
Pentium 4		3060	$586	4.2	$138.68	12.4	$47.12
Pentium 4 HT		2400	$265	3.4	$78.44	9.8	$27.11
Pentium 4 HT		2667	$320	3.7	$85.38	10.8	$29.51
Pentium 4 HT		2800	$405	3.9	$102.82	11.4	$35.53
Pentium 4 HT		3060	$603	**4.3**	$140.17	12.4	$48.44
Pentium 4 HT		3200	$920	4.5	$204.59	13.0	$70.70

Table E.1: CPU price/performance
Benchmark results that were directly gathered are shown in **bold type**. Benchmark results that were obtained from the distributed.net database are shown in *italics*.

Appendix F

CD contents

The CD attached to this thesis contains the following directory structure and files:

thesis.pdf	This thesis in PDF format
data/	The Excel spreadsheets used to perform the analysis
ks1/	VHDL source code for the initial key search machine and modules
ks1driver/	C source code to control the initial key search machine
ks2des/	VHDL source code for the revised key search machine using the DES cipher module
ks2rc5/	VHDL source code for the revised key search machine using the RC5 cipher module
ks2driver/	C/C++ source code to control the revised key search machine
stat/	C source code that generates RAM initialisation values for the statistical comparator

Bibliography

[1] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. John Wiley & Sons, Inc., January 1996.

[2] M. Blaze, W. Diffie, R. L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener, "Minimal key lengths for symmetric ciphers to provide adequate commercial security," A Report by an Ad Hoc Group of Cryptographers and Computer Scientists, January 1996. [Online]. Available: http://www.schneier.com/paper-keylength.pdf

[3] J. O. Grabbe, "The DES algorithm illustrated," in *Laissez Faire City Times*, vol. 2, no. 28. [Online]. Available: http://www.aci.net/kalliste/des.htm

[4] J. J. G. Savard. A cryptographic compendium. [Online]. Available: http://home.ecn.ab.ca/~jsavard/crypto/intro.htm

[5] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," in *Lecture Notes in Computer Science*, no. 809. Springer-Verlag, 1994, pp. 191–204. [Online]. Available: http://www.schneier.com/paper-blowfish-fse.html

[6] R. L. Rivest, "The RC5 encryption algorithm," in *Practical Cryptography for Data Internetworks*, W. Stallings, Ed. IEEE Computer Society Press, 1996.

[7] J. Keller and B. Seitz, "A hardware-based attack on the A5/1 stream cipher," in *APC 2001*. VDE Verlag, 2001, pp. 155–158. [Online]. Available: http://www.informatik.fernuni-hagen.de/ti2/papers/apc2001-final.pdf

[8] National Security Agency. (1998, May) Skipjack and KEA algorithm specifications. [Online]. Available: http://csrc.nist.gov/encryption/skipjack/skipjack.pdf

[9] E. Biham, "A fast new DES implementation in software," *Lecture Notes in Computer Science*, vol. 1267, pp. 260–??, 1997. [Online]. Available: http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/1997/CS/CS08%91.ps.gz

[10] Electronic Frontier Foundation, *Cracking DES*. O'Reilly, 1998.

[11] Xilinx, Inc., "Virtex-II Pro complete data sheet," September 2003, http://direct.xilinx.com/bvdocs/publications/ds083.pdf.

[12] P. Leong, M. Leong, O. Cheung, T. Tung, C. Kwok, M. Wong, and K. Lee, "Pilchard - a reconfigurable computing platform with memory slot interface," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2001. [Online]. Available: http://www.cse.cuhk.edu.hk/~phwl/papers/pilchard_fccm01.pdf

[13] W. Diffie and M. E. Hellman, "Exhaustive cryptanalysis of the NBS data encryption standard," in *Computer*, June 1977, vol. 10, no. 6, pp. 74–84.

[14] R. McLaughlin, "Yet another machine to break DES," *Cryptologia*, vol. 16, no. 2, pp. 136–144, April 1992.

[15] M. J. Wiener, "Efficient DES key search," in *Practical Cryptography for Data Internetworks*, W. Stallings, Ed. IEEE Computer Society Press, 1996, pp. 31–79.

[16] I. Goldberg and D. Wagner, "Architectural considerations for cryptanalytic hardware," CS252 Report, 1996. [Online]. Available: http://www.cs.berkeley.edu/~iang/isaac/hardware/paper.ps

[17] M. J. Wiener, "Efficient DES key search - an update," in *Cryptobytes*, RSA Laboratories, Ed., 1997, vol. 3, no. 2, pp. 6–8. [Online]. Available: ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto3n2.pdf

[18] J.-P. Kaps and C. Paar, "Fast DES implementation for FPGAs and its application to a universal key-search machine," in *Selected Areas in Cryptography*, 1998, pp. 234–247.

[19] I. Hamer and P. Chow, "DES cracking on the Transmogrifier 2a," in *Lecture Notes in Computer Science*, ser. Cryptographic Hardware and Embedded Systems. Springer-Verlag, 1999, no. 1717, pp. 13–24. [Online]. Available: http://www.eecg.toronto.edu/~pc/research/publications/des.ches99.ps.gz

[20] P. D. Kundarewich, S. J. Wilton, and A. J. Hu, "A cpld-based rc4 cracking system," in *Canadian Conference on Electrical and Computer Engineering*, 1999. [Online]. Available: http://www.ee.ubc.ca/~stevew/papers/pdf/ccece99.pdf

[21] K. L. K.H. Tsoi and P. Leong, "A massively parallel RC4 key search engine," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2002, pp. 13–21. [Online]. Available: http://www.cse.cuhk.edu.hk/~phwl/papers/vrvw_fccm02.pdf

[22] P. Kocher, "Breaking DES," in *Cryptobytes*, RSA Laboratories, Ed., 1999, vol. 4, no. 2, pp. 1–5. [Online]. Available: ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto4n2.pdf

[23] M. Blaze. (1997, June) A better DES challenge. [Online]. Available: http://www.privacy.nb.ca/cryptography/archives/cryptography/html/1997-0%6/0127.html

[24] R. Clayton and M. Bond. Experience using a low-cost fpga design to crack des keys. [Online]. Available: http://www.cl.cam.ac.uk/users/rnc1/descrack/DEScracker.html

[25] T. Pornin and J. Stern, "Software-hardware trade-offs; application to A5/1 cryptanalysis," in *Lecture Notes in Computer Science*, ser. CHES 99. Springer-Verlag, 2000, pp. 318–327. [Online]. Available: http://www.di.ens.fr/~stern/data/St91.pdf

[26] (2003, October) distributed.net: Node Zero. [Online]. Available: http://www.distributed.net/

[27] C. M. Curtin. (1998, June) DESCHALL. [Online]. Available: http://www.interhack.net/projects/deschall/

[28] (1997, May) SolNET DES Challenge Attack. [Online]. Available: http://www.des.sollentuna.se/

[29] The RSA Laboratories Secret-Key Challenge. RSA Security. [Online]. Available: http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html

[30] (2003, October) distributed.net: Project CSC. [Online]. Available: http://www.distributed.net/csc/

[31] (1997, January) 40-bit crypto proves no problem. [Online]. Available: http://news.com.com/2100-1017-266268.html?legacy=cnet

[32] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," in *IEEE Transactions on VLSI Systems*, ser. IEEE Transactions on VLSI Systems, August 2001, vol. 9, no. 4.

[33] J. Leonard and W. H. Mangione-Smith, "A case study of partially evaluated hardware circuits: Key-specific DES," in *Field-Programmable Logic and Applications. 7th International Workshop*, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds., vol. 1304. London, U.K.: Springer-Verlag, 1997, pp. 151–160. [Online]. Available: citeseer.nj.nec.com/leonard97case.html

[34] D. Wagner and S. M. Bellovin, "A programmable plaintext recognizer," 1994. [Online]. Available: ftp://ftp.research.att.com/dist/smb/recog.ps

[35] C. Eilbeck. My crypto page. [Online]. Available: http://www.yordas.demon.co.uk/crypto/

[36] Xilinx, Inc. SRL16 16-bit shift register look-up-table (LUT). [Online]. Available: http://toolbox.xilinx.com/docsan/xilinx5/data/docs/lib/lib0393_377.html

[37] E. Soha. (1998, May) RC5 on FPGAs. No longer available from original source. [Online]. Available: http://web.archive.org/web/19981205053422/http://www-inst.eecs.berkeley%.edu/~barrel/rc5.html

[38] P. Campbell. An RC5-64 Crack Engine. [Online]. Available: http://www.taniwha.com/~paul/rc5.html

[39] P. Alfke and B. New, "Multiplexers and barrel shifters in XC3000/XC3100," Xilinx, Inc., Tech. Rep. [Online]. Available: http://direct.xilinx.com/bvdocs/appnotes/xapp026.pdf

[40] Xilinx, Inc., "XC4000E and XC4000X Series Field Programmable Gate Arrays," May 1999. [Online]. Available: http://www.xilinx.com/bvdocs/publications/4000.pdf

[41] Xilinx Inc., "Virtex-E 1.8V Field Programmable Gate Arrays," July 2002. [Online]. Available: http://direct.xilinx.com/bvdocs/publications/ds022.pdf

[42] distributed.net. (2003, October) distributed.net: Client Speed Comparisons. [Online]. Available: http://n0cgi.distributed.net/speed/

[43] (1997, May) SolNET DES Challenge Attack: Download Page. [Online]. Available: http://www.des.sollentuna.se/download.html

[44] A. Biryukov, A. Shamir, and D. Wagner, "Real time cryptanalysis of A5/1 on a PC," *Lecture Notes in Computer Science*, vol. 1978, pp. 1+, 2001.

[45] (2003, October) Avnet electronics marketing. [Online]. Available: http://em.avnet.com/

[46] E. Peltzer, October 2003, private communication.

[47] S. Satria, October 2003, private communication.

[48] C. D. Ulmer, "Configurable Computing: Practical Use of Field Programmable Gate Arrays," Ph.D. dissertation, School of Electrical and Computer Engineering, Georgia Institute of Technology, January 1999. [Online]. Available: http://users.ece.gatech.edu/~grimace/research/reports/qual_report.pdf

[49] "Gate count capacity metrics for FPGAs," Feb 1997. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp059.pdf

[50] NEC Electronics America, Inc. FPGA to ASIC Conversion. [Online]. Available: http://www.necelam.com/asic/conversion.cfm

[51] NEC Electronics. NEC: Gate array information. [Online]. Available: http://www.necgatearray.com/content.nsf/webpages/gatearrayinfo

[52] The MOSIS Service. MOSIS Integrated Circuit Fabrication Service. [Online]. Available: http://www.mosis.org/

[53] The MOSIS Service. Domestic Price List for MOSIS IC Prototyping Service. [Online]. Available: http://www.mosis.org/Orders/Prices/price-list-domestic.html#tsmc25_logi%c

[54] The MOSIS Service. MOSIS Domestic Price List for ASAT Plastic Packages. [Online]. Available: http://www.mosis.org/products/assembly/plastic/price_domestic_asat.html

[55] distributed.net. RC5-72 Live Stats. [Online]. Available: http://www1.distributed.net/~pstadt/rc5-72/

[56] G. Pure and G. Taylor. The Australian cryptography FAQ. [Online]. Available: http://www.efa.org.au/Issues/Crypto/cryptfaq.html

[57] B.-J. Koops. Crypto law survey. [Online]. Available: http://rechten.uvt.nl/koops/cryptolaw/cls2.htm

[58] Electronic Frontiers Australia Inc. Crypto politics. [Online]. Available: http://www.efa.org.au/Issues/Crypto/crypto2.html

[59] Defence Signals Directorate. Defence Signals Directorate. [Online]. Available: http://www.dsd.gov.au/

[60] Department of Defence. Defence and strategic goods list. [Online]. Available: http://www.defence.gov.au/dmo/id/export/DSGL_2003.pdf

[61] Department of Defence. Export application. [Online]. Available: http://www.defence.gov.au/dmo/id/export/dsec/AC717_Oct_03.pdf

[62] Department of Defense. Application for one time review. [Online]. Available: http://www.defence.gov.au/dmo/id/export/dsec/Onetime.pdf

[63] U. S. Bureau of Industry and Security. HPC - CTP Chart. [Online]. Available: http://www.bxa.doc.gov/HPCs/ctpchart.htm

[64] D. E. Sanger. Letting the Chips Fall Where They May. [Online]. Available: http://www.nytimes.com/library/review/061399china-chips-review.html

[65] Deputy Press Secretary. President changes export controls on computers. [Online]. Available: http://www.whitehouse.gov/news/releases/2002/01/20020102-3.html

[66] Actel Corporation. Actel: Products & Services: Antifuse Devices: Axcelerator. [Online]. Available: http://www.actel.com/products/axcelerator/index.html