Architectural considerations for cryptanalytic hardware

Ian Goldberg and David Wagner
{iang,daw}@cs.berkeley.edu

Abstract

We examine issues in high-performance cryptanalysis, focusing on the use of programmable logic. Several standard techniques from computer architecture are adapted and applied to this application. We present performance measurements for RC4, A5, DES, and CDMF; these measurements were taken from actual implementations. We conclude by estimating the resources needed to break these encryption algorithms.

1 Introduction

Large-scale open electronic communications networks are spreading: for example, mobile computing is on the rise, the Internet is experiencing exponential growth, and electronic commerice is a hot topic. With these advances comes a need for robust security mechanisms, and they in turn depend critically on cryptographic protection. At the same time, computer power has been growing at dizzying rates, matching or exceeding Moore's Law. Therefore, in this rapidly changing environment, it is important to assess the strength of deployed encryption algorithms against the tremendous computational power available to potential adversaries.

The best attacks on today's symmetric-key encryption algorithms simply apply massive computing resources to break their security by pure brute force. If a cryptographic algorithm is secure, it will be far too expensive for an attacker to gather the processing power necessary for such a brute-force cryptanalytic attack to succeed. Assessing the security of a cryptographic algorithm against this threat, then, involves surveying the state of the art in cryptanalytic computational power and estimating the investment required to mount this type of attack.

This paper explores the use of programmable logic hardware devices in cryptanalytic applications. Programmable logic attempts to provide much of the premier performance available from custom hardware, while partially retaining the reconfigurability and ease of development benefits found in software.

Our research draws heavily on the computer architecture field. Surprisingly, many techniques, tools, and models for the design of general-purpose processors also proved useful in the specialized domain of cryptanalytic hardware. We investigate the benefits of various forms of parallelism, including pipelining and superscalar architectures. We also examine and identify critical structural hazards and data hazards, as well as the crucial performance bottlenecks. This paper focuses especially on an analogue of the central "CPU time" formula from [20]. By framing the problem from the perspective of system architects, we were able to take advantage of the extensive knowledge base available in the architecture literature.

This paper is organized as follows. Section 2 elaborates on the need for estimates of the performance of cryptanalytic hardware, and Section 3 lists previous work which touches on this project and influenced our approach. Next, Section 4 introduces our experimental methodology and goals. Section 5 describes our design, implementation, and data in depth, providing a detailed technical analysis. Finally, Section 6 briefly identifies some areas for future research, and Section 7 concludes the paper.

2 Motivation

There is currently a strong need for a solid assessment of the resources required to break the common cryptographic algorithms. This information is a crucial data point for system designers—they need this information to determine which encryption algorithm is appropriate for their system. The need is only intensifying: weak encryption is becoming the norm, earlier assessments are either incomplete or out-of-date, and steady increases in computing power are threatening the viability of these weak encryption systems.

Security is little more than economics. A cryptographic system is secure when it costs more to break it than the data it is protecting is worth. Accordingly, determining the strength of an encryption algorithm comes down to measuring the cost of the cryptanalytic resources needed to break the system. That explains the basic need for an evaluation of the cryptanalytic performance possible today.

In fact, several recent factors make the need more urgent. Weak encryption is being widely deployed. SSL with 40-bit RC4 is becoming a de facto standard for secure Web channels, largely because of Netscape's support. GSM, a European mobile telephony system, depends for its link-layer security on A5, an apparently weakened algorithm. Export restrictions are largely to blame for the recent preponderance of weak encryption algorithms; they are an unfortunate fact of life at the moment. This intensifies the need for accurate estimates of the true protection these cryptographic algorithms offer. For extremely strong algorithms, it is sufficient to provide order-of-magnitude estimates to show that breaking these algorithms requires absurd collections of resources; but when it is feasible (or barely feasible) to break an encryption algorithm, it becomes extremely important to pinpoint the cost of cryptanalysis accurately.

Section 3 lists several earlier algorithm assessments. DES has received by far the most attention, but we are also greatly interested in the (today all-too-common) case of exportable encryption algorithms. Most of the experience with weak encryption systems has been with software cryptanalysis; yet programmable logic may be the most cost-effective method of assembling computational power for this problem. A recent paper [4] did briefly address the cost-effectiveness of programmable logic, but their estimate appears to be based on flawed assumptions. The one work which investigated the problem most closely [22] was a good start, but it didn't go far enough: their estimates were based on theoretical calculations, instead of real implementations and measurements.

Therefore, there is new ground to cover, and previous work to validate. We will explore the applicability and performance of programmable logic to cryptanalysis of A5, DES, CDMF, and RC4. This paper attempts to provide a solid, rigorous assessment of the economics of cryptanalysis, relying on actual implementations and experimental measurements.

3 Related work

Previous exploration into exhaustive keysearch has tended to concentrate on either software implementations or custom hardware designs; not much has been reported on FPGA (programmable logic) architectures. We will survey the results available in the open literature.

The first public brute-force cryptanalysis of 40-bit exportable RC4 appeared from the Internet `cypherpunks` community. (The NSA (National Security Agency) had almost certainly mounted an exhaustive 40-bit search of RC4 long before that, but they're playing their cards close to their chest.) The `cypherpunks` are a loose-knit community dedicated to exploring the social ramifications of cryptography. To demonstrate the need for more secure encryption, Hal Finney challenged his fellow `cypherpunks` to break 40-bit RC4 [16]. Soon Adam Back, David Byers, and Eric Young announced [3] that they had successfully searched the 40-bit keyspace with a software implementation running on the idle cycles of several workstations. At the same time, Damien Doligez had also independently finished a succesful sweep of the RC4 40-bit keys [12], with the same software implementation. Not long later, Piete Brooks, Adam Back, Andrew Roos, and Andy Brown organized a distributed effort [5] which used donated idle cycles from many machines across the Internet to finish a second challenge in 31 hours, again using a similar software implementation. The `cypherpunks` efforts gave us a fairly accurate estimate of the complexity of exhaustively searching the RC4 40-bit keyspace in software.

There have been no reports of any experience with exhaustive keysearch of A5 in the open literature. The details of the A5 algorithm were only recently revealed to the public [1], so it is perhaps not surprising that it has received less attention. Several cryptographers' initial reaction was that there must be a trivial brute-force attack on A5 requiring 2^{40} operations [26, 1]. No such attack ever materialized, and it became clear that the matter was not so trivial as initially imagined [26, 2]. The current consensus appears to be that A5's strength is possibly somewhat more than a 40-bit cipher but less than its 64-bit key might indicate.

There have not been any reports on CDMF exhaustive keysearch in the literature, either. On the other hand, CDMF is very similar to DES—it is essentially DES with a reduced 40-bit keylength—so all the research into understanding DES keysearch will apply immediately to CDMF. As we shall see, there has been extensive work examining DES brute-force cryptanalysis.

There have been many studies into the economics of a DES keysearch implementation in custom hardware. (No one has seriously proposed breaking DES via software, as general-purpose computers are orders of magnitude slower at this task than specialized hardware.) The earliest estimate came not long after DES was ratified as a national standard. Whit Diffie and Martin Hellman designed a system containing a large number of custom-designed chips [11]. They estimated that their \$20 million architecture could recover a DES key each day. After their paper appeared, great controversy ensued. Some argued that the mean time between failures would be inherently so small that the machine could never work; Diffie and Hellman refuted these objections, although they also increased their cost estimate somewhat [27, p. 283]. After the controversy died down, the final estimate was that DES would be insecure by the year 1990 [19]. A later paper suggested that a \$1 million custom-designed hardware architecture could break DES in 9 days with technology forecasted to be available by 1995 [18]. Another more recent estimate took advantage of an extremely fast DES chip (designed for normal cryptographic use, not cryptanalysis), concluding that a \$1 million assembly could search the DES key space in 8 days [31, 13, 14]. Yet another study examined the feasibility of using existing general-purpose content-addressable processors, and concluded that a DES keysearch would take 30 days on them with a \$1 million investment [30]. Even more writing on the subject of hardware DES keysearch can be found in [25], and some issues in DES chip design can be found in [21, 15, 6].

All these estimates were superseded by a compelling 1993 paper [31] from Michael Wiener. He went to the effort of assembling a very comprehensive design (extending for a hefty 42 pages!) of a custom-hardware DES keysearch machine, including low-level chip schematics as well as detailed plans for controllers and shelving. After a \$0.5 million investment to design the machine and \$1 million to build it, a DES key could be recovered each 3.5 hours, he argued. (Note the large development cost. This is a unique attribute of custom hardware designs.) His work has remained the definitive estimate of DES keysearch cost since then. On the other hand, we have seen 3 years of steady progress in chip performance and cost since then, and Moore's law remains as true as ever, so Wiener's figures should be adjusted downward accordingly.

This year an ad-hoc group of experts was convened to recommend appropriate cryptographic key lengths for corporate security; their report [4] was very influential. In this larger context, they very briefly surveyed the application of software, reconfigurable logic, and custom hardware to the brute-force cryptanalysis of 40-bit RC4 and (56-bit) DES. We are a bit skeptical about the precise performance predicted for an RC4-cracking chip: they claimed that a single \$400 FPGA ought to be able to recover a 40-bit RC4 key in five hours. (Amortizing this over many keysearchs, they determined that each keysearch would cost \$0.08, causing some to refer to 40-bit RC4 as "8-cent encryption".) This estimate seems extremely optimistic, as it would require 30 million key trials per second; RC4 key setup requires at least 1024 serialized operations (256 iterations of a loop, with 4 memory accesses and calculations per iteration), so this would represent a throughput of 30 billion operations per second. Even with a dozen parallel independent keysearch engines operating on the chip (which would require serious hardware resources), this would imply clock rates measured in Gigahertz—a rather unlikely scenario! Accordingly, our skepticism helped motivate us to attempt an independent investigation of these issues.

At the other extreme, we are also concerned about gross overestimates of the security of RC4. After several `cypherpunks` folks demonstrated how easy it is to cryptanalyze RC4 with the idle cycles of general-purpose computers, Netscape had to respond. Their note made several good points—for instance, that export controls were to blame, leaving them no choice but to use weak encryption—but their estimate of the cost of breaking

40-bit RC4 was greatly flawed. The first successful keysearch used idle cycles on 120 workstations for 8 days. Netscape claimed that this was $10,000 worth of computing power, concluding that messages worth less than $10,000 can be safely protected with 40-bit RC4 encryption [9]. Exposing the invalidity of this estimate was another motivating force for us.

One unpublished work [22] has studied in depth the relevance of reconfigurable logic to cryptologic applications. They assessed the complexity of a keysearch of DES and RC4 (as well as many other non-cryptanalytic problems). The main weakness of this aspect of their survey is that several of the estimates relied on theoretical predictions instead of real implementations and experimental measurements. In this paper, we attempt to give more rigorous estimates, paying attention to the architectural and economic issues facing these cryptanalytic applications.

4 Technical Approach

4.1 Workloads and architectures

As we have explained earlier, there is much interest in the security of cryptographic algorithms. The algorithms with short keys (such as A5, RC4, CDMF, and DES) are the most interesting to examine, as their security depends intimately on the state-of-the-art in high-performance computing. Therefore, we concentrate on algorithms to break A5, RC4, CDMF, and DES.

Software implementations running on general-purpose microcomputers have received perhaps the most attention [3, 12, 5, 10, 4]. To achieve maximum performance, though, we must also consider the tradeoffs associated with customizable hardware. We will focus mainly on hardware implementations of cryptanalytic algorithms; we then compare the tradeoffs between the hardware and software approaches.

The most specialized approach involves using ASICs: custom-designed hardware, specially tailored to one particular cryptanalytic application. They require a significant initial investment for design and testing; they also must be produced in mass quantity for them to be economical. Therefore, while probably the most efficient approach for a dedicated cryptanalytic application, ASICs require such a large investment that they are probably only of interest to small governments or large corporations—they are certainly not within reach for a class project!

Fortunately, there is a middle ground between ASICs and software. CPLDs (Complex Programmable Logic Devices) provide reconfigurable logic; they are commercially available at low prices. They provide the performance benefits of customizable hardware in small volume at a more reasonable price. We obtained access to a set of Altera FLEX8000 series programmable logic devices—more specifically, 81188GC232 chips. (We greatly appreciate the kind support of Bruce Koball and Eric Hughes!) These are mounted on a RIPP10 board, which can accomodate up to eight FLEX8000 chips and four 128 KB SRAM memory chips.

Therefore, the primary platform of interest was the RIPP10 board with FLEX8000 chips; for comparison purposes, we also investigated several other programmable logic devices, as well as software-driven implementations. The workload consisted of brute-force cryptanalytic applications for RC4, A5, DES, and CDMF.

4.2 The figure of merit

It is important to keep in mind what quantities we are trying to measure. Regardless of whether the methodology involves real implementations or synthetic simulations, the ultimate figure of merit is the performance-cost ratio.

Why is the performance-cost ratio the relevant quantity? In general, our cryptanalytic applications are characterized by extreme suitability to parallelization: the process of exhaustive search over many keys can be broken into many independent small computations without penalty. One fast machine will finish the computation in exactly the same time as two machines which are twice as slow. Therefore, the relevant criterion is the "bang-to-buck" ratio, or more precisely, the numbers of trial keys searched per second per dollar.

4.3 Methodology

We used several methods to understand the architectural tradeoffs and their effect on cryptanalytic applications. We first implemented a few sample cryptanalytic algorithms and directly measured their performance on real workloads and actual architectures. Direct measurement is obviously the most desirable experimental technique; unfortunately, we do not have access to every system in existence. Therefore, to forecast the behavior on other platforms, we also used several simulation tools. In both cases, we examine actual applications and real systems.

4.3.1 Direct measurement

Doing direct measurements on real systems running real applications is conceptually straightforward (but still labor-intensive in practice!). First, we directly implemented the relevant cryptanalytic algorithms for the Altera FLEX8000 platform. Once this is done, it is easy to do several small time trials to measure performance. Finally, we used technical data sheets [8] and price lists [7, 24] from Altera to assess the cost of the system.

We also implemented the applications in software. Measuring performance is easy; fixing a price on the computation is a bit less straightforward, and we will address that in a later section.

4.3.2 Simulations

It would be valuable to obtain measurements for a variety of CPLD architectures. As we only have access to the Altera RIPP10 board and FLEX8000 81188GC232 chips, the experimental procedure becomes a bit more involved. Fortunately, our development environment offers compilation, simulation, and timing analysis tools for several programmable logic devices. We therefore compiled the applications for several other chips and calculated predicted performance estimates with the simulation tools.

An important step for any simulation technique is to validate the simulation process. Accordingly, we applied the same simulation and timing analysis procedure to our applications for the FLEX8000 81188GC232; comparing the performance estimates from the simulation with the direct measurements lets us validate our experimental methodology.

5 Design and Analysis

5.1 Overview

We begin by setting up a model for analysis and describing several design issues that are common to all cryptanalytic hardware.

For this project, we are assuming the "known plaintext" model of cryptanalysis. In this model, an adversary has an encrypted message (the *ciphertext*), and also a small amount of the original message (the *known plaintext*). He also knows what part of the ciphertext corresponds to the known plaintext. The goal of the adversary is to determine the key necessary to decrypt the ciphertext into the known plaintext. He can then use this key to decrypt the rest of the encrypted message.

Other models of cryptanalysis, such as "ciphertext only" or "probabilistic plaintext" [29] are more complicated to use, but do not require an adversary to have specific knowledge of part of the original message. However, as most messages have some well-known parts (a `From` header in a mail message, for example), the known plaintext model turns out to be applicable to almost all situations.

For a cryptographic algorithm to be considered secure, there must be no way to determine the decryption key which is faster than just trying every possible key, and seeing which one works (note that this is a necessary, but not sufficient, condition). This method is called *brute force*.

Breaking a cryptographic algorithm by brute force involves the following steps:

For each key in the keyspace

- Perform key setup
- Decrypt the ciphertext and compare it to the known plaintext

As will be seen below, different algorithms spend different amounts of time in the two steps. (For instance, *stream ciphers*—which generate output one bit at a time—allow us to prune incorrect key guesses very rapidly—while *block ciphers*—which operate on a block at a time—require us to generate the entire output block before any comparison is possible. DES and CDMF are block ciphers; A5 and RC4 are stream ciphers.)

We measure the expected number of cycles for each of the two steps for each key, and add them to determine a *Cycles per Key*, or *CPK* value for the algorithm.

Similar to the formula for CPU time found in [20]:

$$\text{CPU time} = \text{Instruction Count x CPI x Clock cycle time}$$

we have a formula for brute-force searching a keyspace:

$$\text{Search time} = \text{Keys to check x CPK x Clock cycle time}$$

As with the [20] equation, we ignore CPU time. This is valid because we take care to avoid I/O as much as possible. Cryptanalytic applications are typically compute-bound, so this is an important optimization.

In the above formula, "Keys to check" indicates the number of keys to search; this can simply be the total number of keys that can be used with the algorithm, or, in the event that many chips are being used to simultaneously search the keyspace, it can be some fraction thereof.

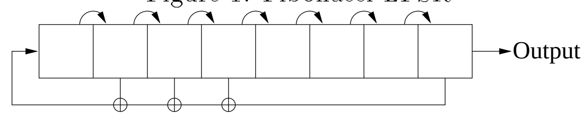
"CPK", as described above, is defined to be "KeySetup + Comparison". "KeySetup" is the number of cycles required to load a key into the algorithm's internal data structures, so that the key search engine is ready to produce output. "Comparison" is the expected number of cycles required for the algorithm to produce enough output so that it can be determined whether the key is the correct one. Note that different algorithms divide their time differently between these two parts, as will be seen in more detail below.

"Clock cycle time" is exactly what one would expect; algorithms that attempt to do more complicated work in one cycle will tend to have a higher clock cycle time. This is also the factor that will vary most when using different models of hardware, as faster (more expensive?) chips have smaller gate delays. One important design feature common to all brute-forcing algorithms also affects this factor: how does one cycle through all of the keys in the keyspace? The obvious solution (to simply start at 0, and increment until the correct key is found) turns out to be a bad one, as incrementing a number of even 8 bits causes unacceptably large gate delays in propagating the carry. Tricks such as carry-save arithmetic [20] are usually not useful here, because keys are usually not used by the encryption algorithms as numbers, but rather, as bit strings.

A better solution [31], which uses the fact that the keys need not be checked in sequential order, is to use a *linear feedback shift register* [27], or *LFSR*. An LFSR is a register that can either be *loaded* (to set the register's value), or have its existing value *shifted* (in order to output 1 bit, and to change the register's value). Of the two styles of LFSR, the usual style is called a *Fibonacci LFSR*. To shift a Fibonacci LFSR, simply copy each bit to its neighbor on the right. The original rightmost bit is considered the output. The bit that is shifted in at the left is the parity of some specific subset of the bits (the *taps*) of the register (see Figure 1).

The most important properties of an LFSR are that it has a low (constant) gate delay, and more importantly,

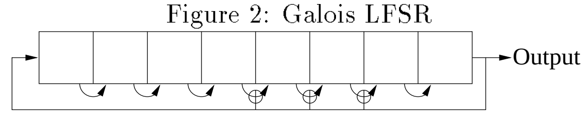
Figure 1: Fibonacci LFSR



if the taps are chosen properly, repeated shifting (starting with any non-zero value) will cycle through every possible non-zero value of the register.

The other style of LFSR is called a *Galois LFSR*, which has the same properties as the Fibonacci LFSR, but is shifted differently. To shift a Galois LFSR, copy each bit to its neighbor on the right, except for the taps, for which the rightmost bit of the register is XOR'd in before the copy is done. The bit that is shifted in at the left is the original rightmost bit, which is also considered the output (see Figure 2).

Figure 2: Galois LFSR



The advantage of a Galois LFSR over a Fibonacci LFSR when being implemented in hardware is that a Galois LFSR usually has an even lower gate delay than a Fibonacci LFSR, resulting in a potentially lower clock cycle time. For this reason, Galois LFSRs are usually used to cycle through the list of possible keys.

In order to take advantage of parallelism, one must be able to distribute the keyspace equitably among the multiple hardware devices. Standard mathematical techniques allow us to easily calculate the value of the shift register after any given number of shifts. From this, we can determine evenly separated starting positions for each device in the search engine.

We will now describe the design issues and analysis that were performed when we implemented various encryption algorithms in programmable logic.

5.2 A5

A5 [1] is the encryption algorithm used in GSM, the European standard for digital cellular telephones. It consists of three Fibonacci LFSRs of sizes 19, 22, and 23 respectively, which are initially loaded with the contents of the 64-bit key. The middle bits of all three LFSRs are examined at each clock cycle to determine which registers shift and which do not (at least two of the three registers shift in each clock cycle). The parity of the high bits of the LFSRs is output after each shift, and this output bitstream is XOR'd with the ciphertext to recover the original message.

This algorithm is quite well-suited for implementation in hardware due to the simplicity of LFSRs; given that it was designed for use in cellular phones, in which limited resources are available, this should not be surprising. The simplicity of the algorithm leaves almost no room for creativity to the implementer.

The resource requirements for A5 are quite minimal; they consist mainly of the 64 flipflops that make up the three LFSRs. In this algorithm, the key setup time is trivial (a single cycle to load the LFSRs with their initial state); the majority of the algorithm consists of comparing the output of the generator (which comes out at a rate of 1 bit per cycle) to the expected output. Since incorrect keys produce essentially random data, the expected number of bits we need to check before rejecting a key is 2. Thus, the total number of cycles per key for A5 is CPK = KeySetup + Comparison = 1 + 2 = 3.

5.3 RC4

RC4 [27] is the encryption algorithm used in, among other things, the Secure Sockets Layer (SSL) protocol [17] used by Netscape and other World Wide Web browsers to transmit encrypted information (such as banking transactions) over the Internet. RC4 is quite a simple algorithm; start with a 256-byte read-only array `K` that stores the key (repeat the key as often as necessary to fill `K`), a 256-byte random-access array `S`, and two 8-bit registers `i` and `j`.

To do key setup, start with `j=0`, and do:

```
for i = 0 to 255:
        S[i] = i
for i = 0 to 255:
        j = (j + S[i] + K[i]) mod 256
        swap S[i] and S[j]
```

Once the key setup is complete, set `i=j=0`, and to generate each byte, do:

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap S[i] and S[j]
output S[(S[i] + S[j]) mod 256]
```

The sequence of bytes outputted is XOR'd with the ciphertext to recover the original message.

SSL, one common system that uses RC4, has a small added complexity. Instead of the key being copied into the array `K`, as described above, it is first processed by the MD5 hash function; the result of the MD5 computation is then copied into `K`. Our design and analysis does not include MD5, which is quite large, complicated, and includes many 32-bit additions, so readers hoping to break SSL should keep in mind that their performance will be substantially worse than that determined below.

The resource requirements for RC4 are considerable. Most notably, it requires 258 bytes of state (compare 8 bytes of state for A5), 256 bytes of which need to be accessed randomly. Such resources were beyond the capabilities of the programmable logic chips we had available, but fortunately the board on which the logic chips were mounted had 128 KB of SRAM accessible to the logic chips via a bus; we stored the array `S` in this SRAM. Note that the key array `K` is accessed in a predictable order, so it was not necessary to store it in the SRAM.

Unfortunately, when trying to produce intstruction-level parallelism in the algorithm, the single port to the SRAM becomes a structural hazard. For this reason, it was necessary to serialize accesses to this SRAM. Initially, we expected that going off-chip to access the SRAM would be the bottleneck that determined the minimum clock cycle time; section 5.5, below, shows that we were incorrect.

We now calculate the "Cycles per Key" value for RC4. Examining the key setup code, it is clear that the first loop requires 1 cycle to initialize `i` to 0, and 256 cycles to complete, and each iteration of the second loop requires 4 cycles (1 each to read and write `S[i]` and `S[j]`), for a total key setup time of 1281 cycles.

Similarly, each byte of output requires 5 cycles to produce (1 each to read and write `S[i]` and `S[j]`, and 1 to read `S[(S[i] + S[j]) mod 256]`. The expected number of bytes needed to determine whether the guessed key is correct is $\left(1 - \frac{1}{256}\right)^{-1} < 1.004$, so the value of "Comparison" is very near 5. Thus we calculate the total Cycles per Key to be CPK = KeySetup + Comparison = 1281 + 5 = 1286.

5.4 DES and CDMF

DES is the national Data Encryption Standard; it enjoys widespread use by the banking industry, as well as being one of the preferred algorithms for securing electronic communications. DES transforms a 64 bit input block into a 64 bit output by a reversible function which depends on the 56 bit key in a highly non-linear way.

The DES algorithm was designed primarily for efficiency in hardware, and thus has several distinguishing features worth noting. It consists of an initial and final permutation and 16 rounds of main processing, with each round transforming the input bits via a "mix-and-mash" process. Bit permutations are used extensively; of course, they are trivial to do in hardware by simply reordering wires. Each round also contains 8 different "Substitution" boxes (or S-boxes for short); the S-boxes are non-linear functions which map 6 input bits to 4 output bits. S-boxes are not very resource-intensive in hardware: they can be implemented as four 6-input boolean functions, and their small size keeps the gate count reasonable. The key is stored in a shift register, rotated before each round, and exclusive-or-ed into the block during each round. This is also straightforward to implement in hardware.

CDMF (Commercial Data Masking Facility) [23] is a related algorithm which uses DES as the underlying transformation; the only difference is that it weakens the key to meet US export restrictions. CDMF has an effective 40-bit keylength, which is then expanded to a 56 bit DES key by using another DES transformation. Loading a CDMF key requires one initial DES operation, and transforming each 64 bit block requires one DES operation. Therefore an implementation of a DES keysearch application leads easily to a CDMF keysearch engine with half the search rate.

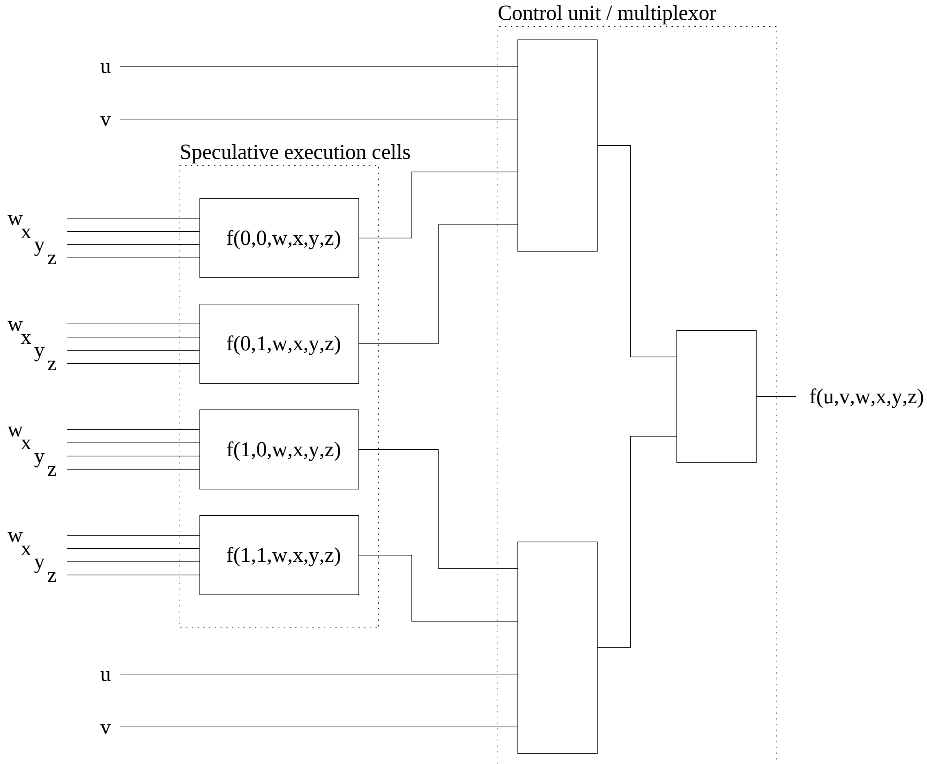
Our DES implementation was forced to be rather minimal to fit in the limited resources available on our chip. We implemented one round of DES, with the appropriate S-boxes and bit permutations. Some extra flip-flops and a state machine allow us to iterate the round function 16 times; there was not sufficient space (i.e. logic gates) available to implement 16 instantiations of each S-box.

The S-boxes are perhaps the most critical component, and we tried several different implementation approaches for them. One natural way to describe each S-box is as a 64-entry lookup table containing 4 bit entries. This might be a good choice if the chip had contained some user-configurable ROM; ours didn't. A similar approach takes advantage of the compiler support for "case" statements, which gets translated into a hardware structure containing a 64-line demultiplexor and or gates expressing the relevant minterms. This structure minimizes gate delay at the expense of space resources. In fact, this structure increased the gate requirements significantly, to the point where the 8 S-boxes alone required more hardware resources than our overworked chip had to offer. The compiler was not particularly helpful at doing space-time tradeoffs to minimize the space requirements, so we ended up optimizing the S-box functions by hand.

The manual optimization we settled on can be viewed as a form of speculative execution. First, note that it suffices to describe how to compute the 6-bit to 1-bit boolean function that calculates one output bit of some S-box. Since the S-boxes behave roughly like they were chosen at random, we don't expect to find any structure in the outputs—i.e. each output will be an uncorrelated non-linear function of the inputs—so this is roughly optimal. To compute such a 6-to-1 function, we first isolate 2 of the 6 input bits as control bits. We do speculative execution with four functional cells; each cell computes the output of the 6-to-1 function under a speculative assumption about the 2 control bits. As there are four possible values of the control bits, the four functional cells enumerate all possibilities. At the same time the functional cells are computing their 4-to-1 function, a multiplexor unit concurrently selects one of the functional cells. The calculation of the 6-to-1 function via speculative execution is depicted in Figure 3. This choice of S-box implementation structure is tailored to our Altera FLEX8000 chips: these chips are organized as an array of logic cells, where each logic cell can compute an arbitrary (configurable) 4-to-1 boolean function. For chips with a different organization, some other manual optimization might be more appropriate.

The "Search time" equation for our CDMF implementation is not hard to analyze. One can easily count the CPK by direct inspection of our implementation. We have a finite state machine with 4 states, labelled from **a** to **d**. The cycle-by-cycle breakdown of the "KeySetup" time for one CDMF encryption is as follows:

Figure 3: Calculation of a boolean function with 6 inputs



a 1 cycle to increment the key and load in the 40-bit CDMF trial key

b 1 cycle to perform the DES input permutation

c 16 cycles to perform 16 rounds of encryption

d 1 cycle to perform the DES final permutation and load in the 64 bit plaintext block

We can see that the "KeySetup" time is 19 cycles. An enumeration of the output generation and comparison stage yields

b 1 cycle to perform the DES input permutation

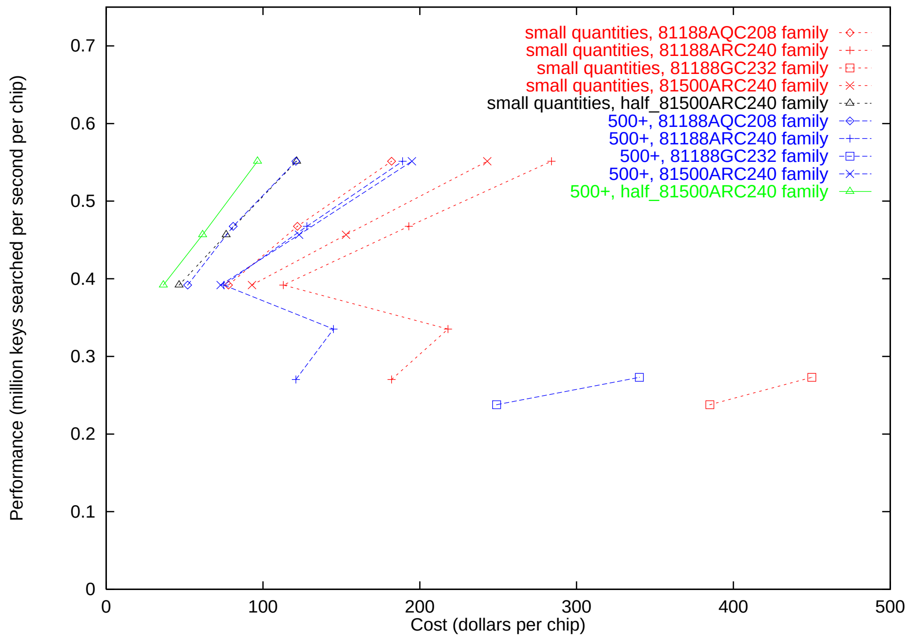
c 16 cycles to perform 16 rounds of encryption

d 1 cycle to perform the DES final permutation, compare the ciphertext block to the expected value, and return to state **a** if this trial key was incorrect

This means that the "Comparison" time is 18 cycles, so the total CPK is $19 + 18 = 37$. Note that DES encrypts the entire 64 bit block at once, unlike a stream cipher, so we check all of the output bits in parallel.

The hardware resources required by CDMF are reasonable but non-negligible for commercial CPLDs. Our minimal implementation required (the equivalent of) roughly 10000 gates. This is certainly within reach for many newer commercial CPLDs, although there are also many older or less expensive CPLDs which cannot handle the requirements. It is important to keep the entire keysearch engine on one chip; otherwise, inter-chip I/O will severely limit performance.

Figure 4: CDMF cryptanalysis economics



5.5 Analysis

We cross-compiled our cryptanalysis implementations for many different Altera CPLDs, and ran a simulation and timing analysis to measure the maximum applicable clock cycle time. The results are plotted in Figure 4 for CDMF, Figure 5 for A5, and Figure 6 for RC4. Some explanation is in order, as there are a lot of data summarized there. The chip specification (e.g. 81188GC232-3) can be dissected as follows: the 81188 refers to the general family, the 232 specifies a 232-pin package, and the -3 refers to the speed grade (lower numbers are faster). The 81500 is the top of the line Altera FLEX8000 device; the 81188 is a bit less powerful. Chips without the "A" designation were fabricated with an older .8 micron process; the "A" indicates chips that were manufactured with a newer, faster .6 micron process. The figure shows throughput graphed against the initial investment required; the chips with the best performance-to-cost (y/x) ratio are the best buy. The prices are taken from a very recent Altera price list [7, 24]. As there are discounts in large quantities, we have plotted price points for small quantities with a red line and for large batches with a blue line.

We also measured the performance for the 81188GC232-3 chip directly—it is the only one we had access to. Our measurements agreed closely with the simulated timing analysis, confirming the validity of our experimental methodology.

Measurements for DES are not listed. Nonetheless, they track the CDMF performance figures very closely. CDMF consists of two DES encryptions—one for key setup, and one for output generation—with very little overhead. The DES keysearch rates can be derived from Figure 4 by simply doubling the CDMF rate. Also, remember that the DES keyspace is 2^{16} times as large. Our data indicate that if one wanted a machine which could perform a DES keysearch in a year on average, it would suffice to spend \$45,000 to buy 600 of the Altera 81500ARC240-4 CPLDs. (This is a very rough estimate, which does not include overhead such as mounting shelves, etc.)

One can note several interesting things from the graph. First, examine the peculiar zig-zag nature of the 81188ARC240 lines. The points are plotted in order of the chip's rated speed grade, from A-6 on the bottom to A-2 on the top. The strange "zag" occurs because the price for a faster A-4 chip drops significantly below the price for the slower A-5. Altera specifies the A-4, A-3, and A-2 as their "preferred" grades for that chip, presumably because there is more sales volume for those speed grades. If you were to build a keysearch engine out of 81188ARC240 chips, you should try to be right at the "hump"—the A-4 speed grade is the best

Figure 5: A5 cryptanalysis economics

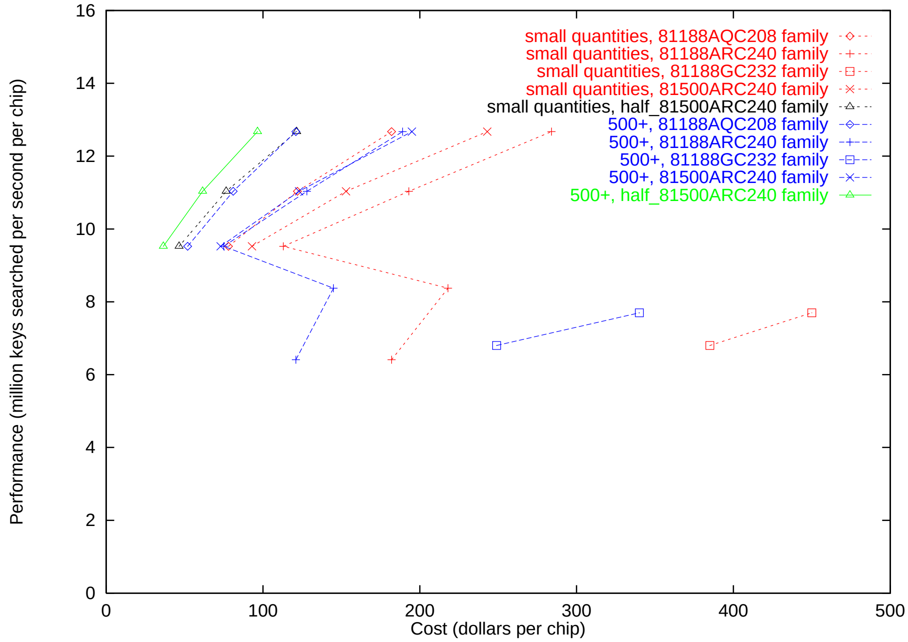
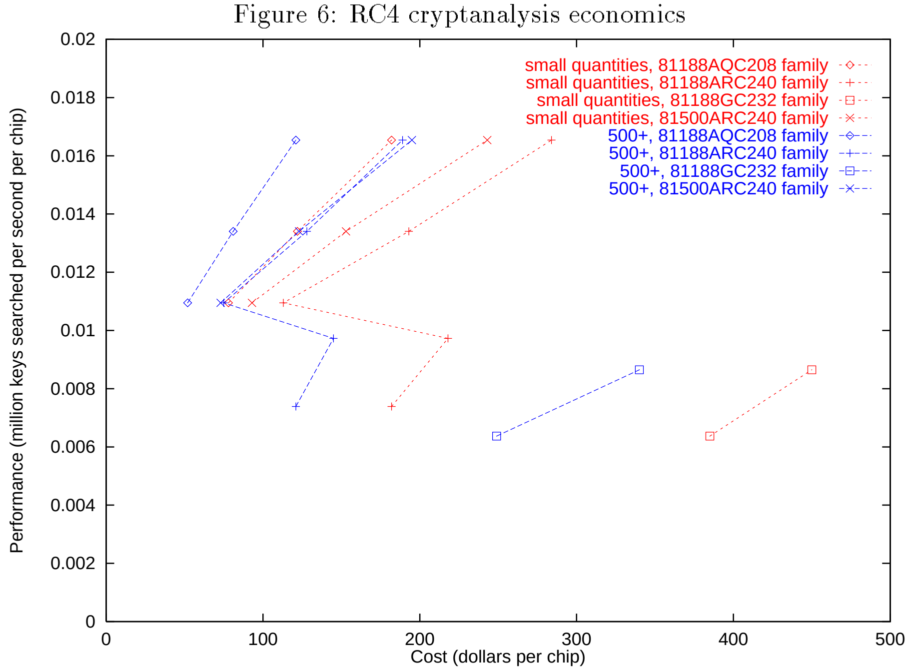


Figure 6: RC4 cryptanalysis economics



buy for that chip.

We have not yet explained the green and black dotted lines. The 81500 line of chips contains more hardware resources than the 81188—1296 instead of 1008 “logic elements”—and this extra space should be taken into account when comparing hardware devices. With our A5 and CDMF implementations, there is quite a bit of space left over on the 81500 chip, as it turns out. Therefore, it is natural to ask whether two independent key trial engines might fit on the same chip. We believe (from close examination of the resource usage) that, with A5 and CDMF, there are sufficient hardware resources on the 81500 to support two superscalar keysearch operations. (It would admittedly be a tight fit.) Because of time pressures, we have not actually implemented this. RC4 requires, it seems, too many resources (mainly flip-flops for internal state) to use this strategy. There would be other difficulties with RC4, anyhow—one would probably need a dual-ported SRAM, or two SRAM chips attached to the CPLD (as discussed below).

One might wonder why we proposed taking advantage of extra hardware resources with a multiple-issue architecture, instead of using (say) advanced pipelining techniques. It is worthwhile to recall why advanced pipelining techniques were developed. On a traditional general-purpose computer, programs are typically serialized so highly that if one were to implement several independent simple processors on the same chip, there simply would not be enough tasks to keep the co-processors busy with useful work. Architects have been blessed with plentiful hardware resources and cursed with the need to speed up single-instruction-stream uniprocessors; this explains the proliferation of sophisticated pipelining methods. (Of course, pipelining does not provide linear speedup with linear increases in hardware resources, like parallelism would, but it is better than nothing!) We are faced with an entirely different situation here. Our cryptanalytic applications encourage virtually unlimited parallelism, so there is no need to look to sophisticated caching schemes for speeds. Achieving parallelism via a superscalar architecture is both simpler and more effective for our purposes.

The projected performance for parallelized 81500 A5 and CDMF keysearch is indicated on the plots with a green and block dotted line, labelled “half_81500ARC240 family”, with the unit price halved to indicate its factor-of-two multiple-issue nature. (We could have doubled the performance instead, but that would have made the graph harder to read, so for ease of comprehension and comparison we chose to halve the cost instead.)

We discussed in class why the future of high-performance computing lies in massively-parallel collections of low-end processors (say, Pentiums), instead of in specialized advanced CPUs. One major reason is that Pentium processors are sold in such large quantities that tremendous economies of scale apply, and specialized processors simply cannot compete with the low-end’s ever-increasing performance-cost ratio. We can see that an analogous situation applies here as well. The graphs show that, for our applications, upgrading to a higher speed grade is almost never worth the increased cost. (Two notable exceptions—the “hump” in the 81188ARC240 plot, and the benefits of using a 81500 with enough hardware resources to implement two keysearch engines on-chip—have already been discussed.) Within each family, the least expensive chip turns out to yield the best performance-to-cost ratio; spending twice as much money on a higher-grade chip in the family never results in twice the performance. On the other hand, upgrading to a more recent “A” designated family—one fabricated with a newer .6 micron process—is a worthwhile move. Altera has listed the “A” chips as their preferred technology, and presumably there is more sales volume for devices on their preferred list (though it might be hard to separate cause from effect here). These charts don’t tell the whole story. Altera is as we write starting to release a new advanced line of reconfigurable logic devices, the FLEX10K architecture. In recommending the 81188 and 81500 devices, we gain extra price-performance benefits by staying a bit behind behind the bleeding edge. Exploiting parallelism with low-end devices is a win for our applications.

We have not yet discussed the impact of software in relation to the hardware performance measurements. Software is a bit trickier to evaluate and compare to the other measurements, as it is not clear how to compare the price of a software solution to a hardware approach. While hardware devices would typically be purchased with one application in mind, often a certain amount of idle cycles on general-purpose computers is available “for free”. Nonetheless, software and hardware approaches typically won’t be in serious competition: the extra expense of hardware is usually not justified until “free” software implementations on general-purpose

Figure 7: Typical software performance on cryptanalytic applications

Algorithm	Keys searched per second
RC4	21900
CDMF	29800
DES	41300
A5	355000

computers are unacceptably slow.

Figure 7 lists the performance of brute-force keysearch applications, as measured on a Pentium P100 machine. Of course these figures will vary widely from computer to computer. For example, we estimate that we could perform a distributed RC4 40-bit keysearch in a weekend or so, and a CDMF 40-bit keysearch in about a night or two, by using idle cycles on the hundreds of general-purpose computers we have access to as Berkeley computer science graduate students.

Many other organizations also have large numbers of computers which are idle much of the time. Many employees and students thus have access to spare computational power which may be harnessed for cryptanalysis, at essentially zero cost. Compare this to Netscape's estimate that amassing enough processing power to break 40-bit RC4 would cost roughly $10,000. For much less than this, one could probably convince a starving graduate student to lend out access to the necessary computer account. In any event, if Netscape were willing to pay $10,000 for the amount of computing power required to break 40-bit RC4, some enterprising student could easily form a extremely profitable business model.

Given a distributed system of general-purpose computers, one can easily compute the maximum rate of 40-bit keysearching possible in idle cycles by assuming that most machines are idle at least half of the time and using estimates such as those in Figure 7; achieving better performance than this calls for hardware. We can see from Figure 7 that our hardware implementations of CDMF, DES, and A5 keysearch are orders of magnitude faster than software; this is not surprising, as these encryption algorithms were designed for efficiency in hardware.

RC4, by contrast, was designed to run efficiently in software, and indeed, as can be seen by comparing Figures 6 and 7, RC4 performs about twice as well in software than on programmable logic. The primary reasons for the large search time on programmable logic are that RC4 has a large "Cycles per Key" value, and a large "Clock cycle time" value: as seen above, the total CPK for the RC4 algorithm is 1286; far larger than the 3 for A5 or the 37 for CDMF. The large clock cycle time stems from the fact that the algorithm contains a number of register additions; as discussed above, these can produce very large gate delays. Unfortunately, changing the additions to LFSRs (as was done above), or using tricks such as carry-save arithmetic, is not appropriate for RC4, as can be seen by examining the algorithm.

Another blow to implementing RC4 efficiently was the particular hardware architecture we had. The programmable logic devices we used were not large enough to store the necessary 256-byte state array on-chip, so we were forced to store them in the external SRAM. However, the algorithm utilizes the SRAM every cycle, so the number of simultaneous RC4 trials we can compute is limited by the number of ports to SRAM that we have available. Unfortunately, on the RIPP10 programming board, not only is the SRAM single-ported, but each SRAM is shared by *two* logic chips. Thus on a fully-populated board with eight logic chips and four SRAMs, we can only perform four simultaneous RC4 trials. Redesigning the programming board to include a port to SRAM for each simultaneous RC4 trial would save some overhead (wasted space on the board), but would not increase the relatively poor performance to cost ratio shown above.

One advantage of software is that the development process is significantly easier. By reusing code (from cryptographic libraries available on the Internet, for example), we prototyped RC4, A5, CDMF, and DES software keysearch applications in a total time of under an hour. In contrast, our programmable logic design and implementation effort took roughly 4 weeks to complete.

Figure 8: Estimating the cost of cryptanalysis: a summary

Algorithm	Investment for average keysearch time of				Architecture components
	1 year	1 week	1 day	1 hour	
RC4	$0	$0	-	-	100 general-purpose computers
CDMF	$0	$0	-	-	100 general-purpose computers
CDMF	$93	$93	$745	$15,000	Altera 81500ARC240-4 CPLDs
DES	$45,000	-	-	-	Altera 81500ARC240-4 CPLDs

Programmable logic has similar advantages over custom-hardware. Development and design would be still more time-consuming and costly for a custom-hardware approach, such as an ASIC. Furthermore, such an ASIC can only be used for one limited algorithm. Programmable logic is more flexible—the hardware devices can be reused for cryptanalysis of many different encryption algorithms with little extra effort. Apparently AccessData, a business that specializes in recovering lost data (i.e. cryptanalysis) for the corporate and law enforcement industries, prefers programmable logic over custom hardware for exactly these reasons [28].

Let us summarize what the charts recommend to one in need of cryptanalytic computational power. RC4 keysearches appear to be most efficiently performed in general-purpose distributed systems. Performing a single isolated 40-bit CDMF keysearch is perhaps best done with distributed software, if time is not of the essence and there are sufficient general-purpose computational resources easily available. For CDMF and A5 keysearch in anything more than that extremely minimal setting, though, reconfigurable logic is the most appropriate solution of the technologies that we examined. Of the devices we surveyed, the Altera 81500ARC240-4 device is the most appropriate and economical choice for cryptanalytic applications; for instance, a $15,000 initial investment buys about 200 of these chips, allowing one to perform on average one CDMF keysearch every hour. The cost scales linearly, requiring approximately 10^8 dollar-seconds for a complete CDMF keysearch; that is, an initial investment of x dollars allows one to search the entire CDMF keyspace in $10^8/x$ seconds, while the average time to find a key is half that. In addition, we provisionally estimate that about $45,000 of CPLD hardware could perform a DES keysearch in a year, as calculated above. Figure 8 summarizes some of these calculations. It takes into account the economies of scale associated with buying many logic devices, and is based on the average-case (not worst-case) search time; the worst-case figure would be twice as large. No figures for A5 are included, because at the moment, there is no consensus among cryptographers as to the size of the keyspace [26].

6 Future work

Due to time and resource limitations, we were only able to examine the Altera FLEX8000 series of programmable logic devices. An obvious extension of this work would be to examine other kinds of devices, such as the new Altera FLEX10K series, or devices from other vendors such as Xylinx. Additionally, it would be worthwhile to examine the technology trends in programmable logic, to determine how they compare to those for general-purpose hardware.

We leave it as an open problem to the reader to actually construct a fully operational DES keysearch engine.

7 Conclusions

We found that RC4 cryptanalysis is most effectively implemented in software. Since RC4 was specifically designed for efficiency on general-purpose computers, it is not entirely surprising that programmable logic fares so poorly. We showed that the estimate in [4] (which inspired the term "8-cent encryption" for 40-bit RC4) is over-optimistic and unrealistic. On the other hand, Netscape's $10,000 estimate was far too large.

Programmable logic devices are very efficient at CDMF cryptanalysis. We estimate that an initial investment of $745 buys enough programmable logic to recover one CDMF key each day; this shows that CDMF is practical to break. Moreover, DES is nearly practical to break; a cryptanalytic engine to do a DES keysearch each year can be built with roughly $45,000 of programmable logic.

Several architectural techniques from the design of general-purpose processors were useful in this project. Adding parallelism, identifying structural and data hazards, identifying performance bottlenecks, and other techniques helped maximize the performance of our design. The cryptanalytic analogue to the "CPU time" equation from [20] was surprisingly useful, lending structure to our analysis.

We also identified several important aspects found only with cryptanalytic applications on programmable logic. In this application, superscalar parallelism is more effective than pipelining. Also, register additions can often be a limiting bottleneck for programmable logic—we avoided them where possible, and suffered large performance hits elsewhere.

By considering architectural issues both common to general-purpose processors and unique to programmable logic, we examined the feasability of using commodity logic devices for cryptanalytic applications.

8 Acknowledgements

This work would not have been possible without the assistance of a number of people. We would like to thank Eric Hughes and Bruce Koball for providing the hardware and software. We would also like to thank Clive McCarthy and Stephen Smith, both of Altera, for their generous support.

9 Availability

This paper, and other related materials, are available on the World Wide Web at

`http://www.cs.berkeley.edu/~iang/isaac/hardware.html`

References

[1] Ross Anderson. A5, June 1994. Post to `sci.crypt` newsgroup. Available on the Internet as `http://chem.leeds.ac.uk/ICAMS/people/jon/a5.html`.

[2] Ross Anderson, April 1996. Personal communication.

[3] Adam Back. Another SSL breakage..., August 1995. Post to `cypherpunks` mailing list. Available on the Internet as `http://dcs.ex.ac.uk/~aba/ssl/`.

[4] Matt Blaze, Whitfield Diffie, Ronald L. Rivest, Bruce Schneier, Tsutomu Shimomura, Eric Thompson, and Michael 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. Available on the Internet as `http://www.bsa.org/policy/encryption/cryptographers.html`.

[5] Piete Brooks. Hal's second challenge, August 1995. Available on the Internet as `http://www.brute.cl.cam.ac.uk/brute/`.

[6] Albert G. Broscius and Jonathan M. Smith. Exploiting parallelism in hardware implementation of the DES. In *Advances in Cryptology: Proceedings of CRYPTO '91*, pages 367–376. Springer-Verlag, 1992.

[7] Altera Corporation. Altera components North America price list, May 1996.

[8] Altera Corporation. Altera home page, 1996. Available on the Internet as `http://www.altera.com/`.

[9] Netscape Communications Corporation. Key challenge, 1995. Available on the Internet as `http://www.netscape.com/newsref/std/key_challenge.html`.

[10] Wei Dai. Speed benchmarks, 1996. Post to `cypherpunks` mailing list. Available on the Internet as `http://www.eskimo.com/~weidai/benchmarks.txt`.

[11] Whitfield Diffie and Martin E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, June 1977.

[12] Damien Doligez. SSL challenge—broken, August 1995. Post to `cypherpunks` mailing list. Available on the Internet as `http://pauillac.inria.fr/~doligez/ssl/`.

[13] H. Eberle and C. P. Thacker. A 1 Gbit/second GaAs DES chip. In *Proceedings of the IEEE 1992 Custom Integrated Circuits Conference*, pages 19.7/1–4. IEEE, May 1992.

[14] Hans Eberle. A high-speed DES implementation for network applications. Technical Report 90, DEC SRC, September 1992.

[15] R. C. Fairfield, A. Matusevich, and J. Plany. An LSI digital encryption processor (DEP). In *Advances in Cryptology: Proceedings of CRYPTO '84*, pages 115–143. Springer-Verlag, 1985.

[16] Hal Finney. SSL RC4 challenge, July 1995. Post to `cypherpunks` mailing list. Available on the Internet as `http://www.portal.com/~hfinney/sslchallong.html`.

[17] A.O. Freier, P. Karlton, and P.C. Kocher. SSL version 3.0, 1995. Internet-Draft `draft-freier-ssl-version3-00.txt`, work in progress.

[18] Gilles Garon and Richard Outerbridge. DES Watch: An examination of the sufficiency of the data encryption standard for financial institution information security in the 1990's. *Cryptologia*, XV(3):177–193, July 1991.

[19] Martin E. Hellman. DES will be totally insecure within ten years. *IEEE Spectrum*, 16(7):32–39, July 1979.

[20] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, 2nd edition, 1996.

[21] Frank Hoornaert, Jo Goubert, and Yvo Desmedt. Efficient hardware implementation of the DES. In *Advances in Cryptology: Proceedings of CRYPTO '84*, pages 147–173. Springer-Verlag, 1985.

[22] Eric Hughes and Bruce Koball. Cryptography and the Altera FLEX 81188, December 1994. Unpublished manuscript.

[23] D.B. Johnson, Sm.M. Matyas, A.V. Le, and J.D. Wilkins. Design of the commercial data masking facility data privacy algorithm. In *1st ACM Conference on Computer and Communications Security*, pages 93–96. ACM Press, 1993.

[24] Clive McCarthy. Personal communication, April 1996.

[25] Robert McLaughlin. Yet another machine to break DES. *Cryptologia*, XVI(2):136–144, April 1992.

[26] Michael Roe, April 1996. Personal communication.

[27] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, New York, 2nd edition, 1994.

[28] Bruce Schneier, April 1996. Personal communication.

[29] David Wagner and Steven M. Bellovin. A probable plaintext recognizer, September 1994. Unpublished manuscript.

[30] Peter C. Wayner. Content-addressable search engines and DES-like systems. In *Advances in Cryptology: Proceedings of CRYPTO '92*, pages 575–586. Springer-Verlag, 1993.

[31] Michael J. Wiener. Efficient DES key search. In *Advances in Cryptology: Proceedings of CRYPTO '93*, Santa Barbara, CA, 1994. Springer.