Appears in the 28th Annual International Symposium on Computer Architecture (ISCA-2001), June 2001

CryptoManiac: A Fast Flexible Architecture for Secure Communication

Lisa Wu, Chris Weaver, and Todd Austin
Advanced Computer Architecture Laboratory
University of Michigan
Ann Arbor, MI 48109
{wul, chriswea, taustin}@eecs.umich.edu

Abstract

The growth of the Internet as a vehicle for secure communication and electronic commerce has brought cryptographic processing performance to the forefront of high throughput system design. This trend will be further underscored with the widespread adoption of secure protocols such as secure IP (IPSEC) and virtual private networks (VPNs).

In this paper, we introduce the CryptoManiac processor, a fast and flexible co-processor for cryptographic workloads. Our design is extremely efficient; we present analysis of a 0.25um physical design that runs the standard Rijndael cipher algorithm 2.25 times faster than a 600MHz Alpha 21264 processor. Moreover, our implementation requires 1/100th the area and power in the same technology. We demonstrate that the performance of our design rivals a state-of-the-art dedicated hardware implementation of the 3DES (triple DES) algorithm, while retaining the flexibility to simultaneously support multiple cipher algorithms. Finally, we define a scalable system architecture that combines CryptoManiac processing elements to exploit inter-session and inter-packet parallelism available in many communication protocols. Using I/O traces and detailed timing simulation, we show that chip multiprocessor configurations can effectively service high throughput applications including secure web and disk I/O processing.

1 Introduction

The widespread adoption of the Internet as a trusted medium for communication and commerce has made cryptography an essential component of modern information systems. The trend towards virtual private networks (VPNs) [15] and secure IP (IPSEC) [3] will further emphasize the significance of cryptographic processing among all types of communication. Security-related processing can consume as much as 95 percent of a server's processing capacity [25]. As demands for secure communication bandwidth grow, efficient cryptographic processing will become increasingly critical to good system performance.

Cryptography is a Greek word that literally means the art of writing secrets [21]. In practice, cryptography is the task of transforming information into a form that is incomprehensible, but at the same time allows the intended recipient to retrieve the original information using a secret key. Cryptographic algorithms (or *ciphers*, as they are often called) are special programs designed to protect sensitive information on public communication networks. During *encryption*, ciphers transform the original *plaintext* message into unintelligible *ciphertext*. *Decryption* is the process of retrieving plaintext from ciphertext. Two forms of cryptography are commonly used in information systems today: *secret-key ciphers* and *public-key ciphers*. Secret-key ciphers (sometimes referred to as symmetric-key ciphers) use a single private key to encrypt and decrypt as illustrated in Figure 1. Public-key ciphers (or asymmetric-key ciphers) use a well-known public key to encrypt and require a different private key to decrypt.

Public-key ciphers have the advantage of being able to establish a secure communication channel without an unsafe exchange of keys. Private-key ciphers, on the other hand, require a shared private key before secure communication can commence. The distribution of the shared private key is the primary obstacle in making symmetric-key ciphers secure. Secret-key ciphers have the advantage of running as much as 1000 times faster than comparable public-key ciphers [30]. To maximize security and performance, most secure protocols use both forms of cryptography. Public key encryption is used at the start of a session to authenticate communicating parties and to securely distribute a shared secret key. The remainder of the session employs efficient secret key algorithms using the private key exchanged during authentication. We refer this type of key management as *public-secret key cryptography.* An example of a system that uses this particular session management strategy is the Secure Sockets Layer (SSL) protocol [38]. SSL is a standard secure protocol that provides secure communication between web servers and web clients. It is supported by most popular web browsers [25].

For very short sessions, fast public-key cipher processing is critical for high transaction throughput. For longer sessions, private-key cipher performance becomes more important. Figure 2 illustrates the SSL run-time breakdown by server processing type. Data shown is collected from a heavily loaded web server running on an iA32 platform [29]. A recent study [2] found that the average size of a web object was 21k bytes. Given that a single session will be composed of many web objects, cryptographic processing will be quickly dominated by private-key cipher execution. As such, the focus of our design effort is on improving private-key cipher performance.

Cryptography can be implemented with software routines, directly in hardware, or a combination of both. A software only approach is the lowest-cost solution but with accordingly lower performance. An example of the hardware-only approach is the IDEA engine [23]. It is targeted specifically at the efficient execution of the IDEA cipher and renders excellent performance. However, its

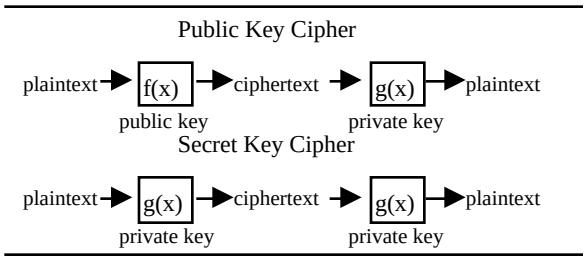

Figure 1. Public-Key and Secret-Key Ciphers.

performance is at the expense of flexibility as the hardware cannot be used for other cryptographic processing tasks.

A hardware/software mixed approach was proposed in our previous study [5] and was shown to achieve an overall performance improvement of 59% over software-only approaches. The technique improved performance of private-key ciphers through instruction set support for fast substitutions, permutations, rotates, and modular arithmetic. In this paper, we further extended this approach through the design of a fast and flexible cryptographic co-processor. Our design, called the *CryptoManiac*, addresses the primary bottleneck in private-key ciphers, namely efficiency, through the application of an efficient VLIW architecture with a well-matched instruction set and functional unit resources. The programmable feature supports many private-key ciphers, in contrast to the IDEA engine. By combining CryptoManiac processors into parallel configurations, we are able to scale cryptographic performance for applications with inter-session and inter-packet parallelism.

We detail the design and implementation of the CryptoManiac processor and analyze its performance using architectural and physical design models. In Section 2, we characterize cipher kernel operations and their various latencies and bottlenecks to gain insight as to how to build an efficient design. In Section 3, we present the CryptoManiac design, detailing the processor architecture, the system architecture, and the instruction set. We then present our experimental framework and methodology, and examine performance results in Section 4. In the final sections, we discuss related work, summarize our findings, and make suggestions for future work.

2 The Nature of Cryptography

Successful cryptographic co-processor design requires a thorough understanding of the nature of private key cryptography, its essential operations, and its inherent bottlenecks. We selected eight cipher kernels for cryptanalysis. They are 3DES [13], Blowfish [11], IDEA [23], Mars [7], RC4 [33], RC6 [32], Rijndael [12], and Twofish [34]. These eight ciphers are selected because each is generally considered a *strong* cipher as they have undergone aggressive review and cryptanalysis. A strong cipher has high resilience to the efforts of an unrelated party attempting to determine the original content of an encrypted message. The two most notable ciphers are 3DES and Rijndael. 3DES runs the US DES [13] standard encryption algorithm. We run the algorithms as specified by the SSL protocol [38]. Rijndael was recently selected as the new US Advanced Encryption Standard (AES) [1]. The rest of the

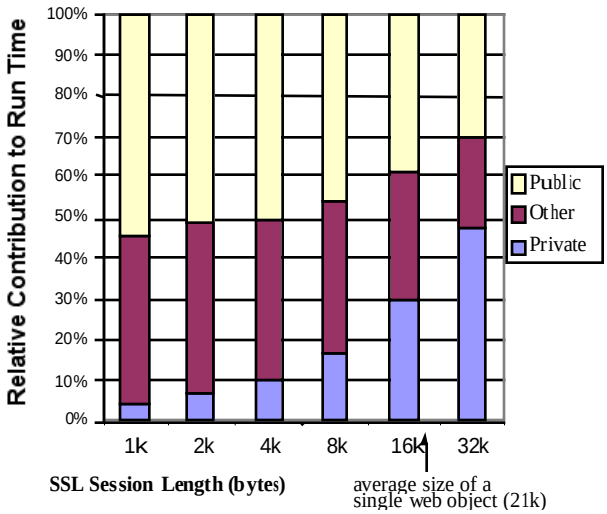

Figure 2. SSL Protocol Relative Contribution to Run Time.

ciphers are either AES second-round candidates or algorithms used in popular software packages.

Cipher algorithms typically have three operational parameters: key size, block size, and number of rounds. The key size is the length of the key used to encrypt or decrypt data. The block size is the amount of data processed each time the cipher kernel is invoked. The number of rounds specifies the total number of iterations executed by the cipher kernel loop. 3DES has a key size of 112 bits, block size of 64 bits, and 48 rounds. Rijndael has key size of 128 bits, block size of 128 bits, and 10 rounds. The remaining kernels use at least 128 bits of key data. The baseline cipher implementations are well tuned for general-purpose machines. Further details on the baseline cipher algorithms can be found in [5].

Figure 3 shows the performance of the cipher kernels executing on a 600MHz Alpha 21264 workstation and an Alpha *dataflow* (DF) machine implemented using the SimpleScalar simulators [5]. Encryption performance is shown in bytes per 1000 clock cycles executed. This is a convenient metric because it represents the encryption performance in MB/s on a 1GHz machine. The dataflow machine is a dynamically scheduled microarchitecture with infinite fetch, decode, execute, and retirement bandwidth. The dataflow machine's performance is never impacted by branch mispredictions, cache misses, or ambiguous store address dependencies. The DF configuration gives the absolute top performance that the original code can achieve.

Many of the ciphers have little parallelism and few bottlenecks. Blowfish, 3DES, IDEA, and RC6 are running within 20% of dataflow machine performance. There is slightly more headroom for Mars and Twofish, with potential speedups of 29% and 76% respectively. RC4 and Rijndael are the outliers, these codes have ample parallelism and could be sped up with more capable hardware.

Figure 4 illustrates the factors that slow down the cipher kernels with performance headroom. The graph shows the performance impact of inserting a single bottleneck into the dataflow machine execution. The resulting performance impact indicates the extent to which the bottleneck is fully

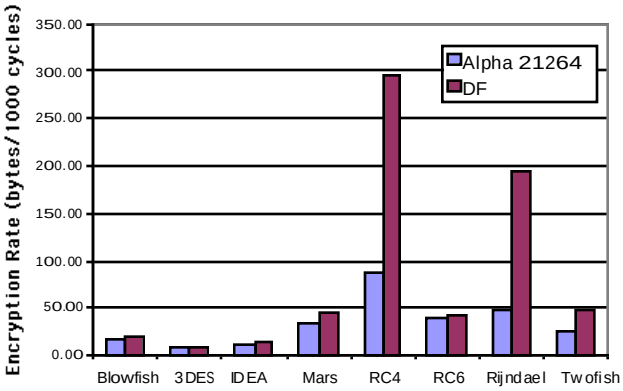

Figure 3. Performance Comparison between Alpha 21264 and Dataflow Machine.

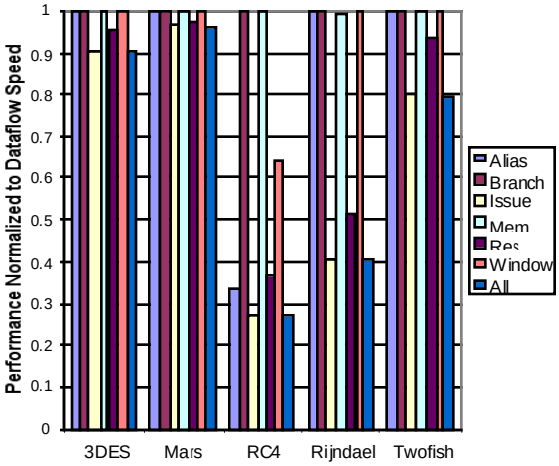

Figure 4. Cipher Kernel Bottleneck Analysis.

exposed during execution, independent of all other bottlenecks. Clearly, branch mispredictions and cache misses are not impediments to good cipher performance. Memory aliases do not cause any slow downs except for RC4, which stores computed values into its substitution tables. The most prevalent factors leading to poor performance are insufficient issue bandwidth and lack of function unit resources. Consequently, ciphers with performance headroom will benefit most from additional resources and issue bandwidth. For the benchmarks running at near dataflow speeds, we will have to rely on latency reduction of individual operations to improve their performance.

To gain a deeper understanding of the cipher kernel operations executed, we profiled each kernel and classified individual operations into nine categories. Their operations are classified into general arithmetic, logical, multiplication, rotate, memory operations, substitutions, permutations, moves, and branches. Figure 5 shows the breakdown by category of all dynamic operations executed. All cipher kernels studied exhibit varied but similar characteristics. Substitution operations implement a key-based transformation function using a byte-indexed array called an "SBOX". Permutation operations rearrange the bit values using a key-parameterized network called an "XBOX". Arithmetic operations are primarily additions and address computations. Multiplication operations include regular multiplication and modular multiplication operations. Logical operations primarily consist of AND and XOR operations. It is interesting to note that very few fundamental operations are used to implement these ciphers. In addition, many of these fundamental operations, e.g. XORs, make inefficient use of the processor clock cycle, creating opportunities for more efficient designs.

To summarize the characterization of these ciphers, they employ few fundamental operations with varied latency. Many of the algorithms have little or no parallelism. The algorithms that do have performance headroom would require more issue bandwidth and function unit resources to execute faster. None of the programs have branch or memory bottlenecks, and most are not slowed down by memory aliases. Given these characteristics, it seems the ideal architecture for fast cryptographic processing is a scalable design that efficiently executes the fundamental operations of cryptography.

3 CryptoManiac Architecture

In order to reach better cipher performance, efficiency becomes the goal of our design. In pursuing our goal, the design will focus on a simple microarchitecture, an efficient implementation of operations, and more efficient use of the clock cycle. CryptoManiac is a 4-wide 32-bit VLIW machine with no cache and a simple branch predictor. Since kernel dependencies through registers and memory are well described, a static VLIW scheduler suffices. The lack of branch bottlenecks eliminates the need for a complex branch predictor. A simple BTB can correctly predict nearly all branches. We chose not to include a cache structure because code and data sets fit comfortably in a small static RAM. We employ a triadic (three input operands) ISA that permits combining of most cryptographic operation pairs for better clock cycle utilization. Finally, the CryptoManiac processing elements can be combined into chip multiprocessor configurations for improved performance on workloads with inter-session and inter-packet parallelism.

3.1 System Architecture

Figure 6 details the high level architecture of the CryptoManiac (CM) processor. The host processor interfaces to the CM through the input (InQ) and output (OutQ) request queues. Cryptographic processing requests are inserted into the InQ by a host processor over a connecting bus. The *request scheduler* distributes host processor requests, in the order received, to CM processing elements. The CM processing elements service requests from the InQ, placing any results produced in the OutQ for the host processor. We envision that the input and output queues would be accessible by the host processor via a standard bus interface, such as a PCI bus. It is sufficient to have one input queue for many CM processing elements, as the computationally intensive nature of cryptographic

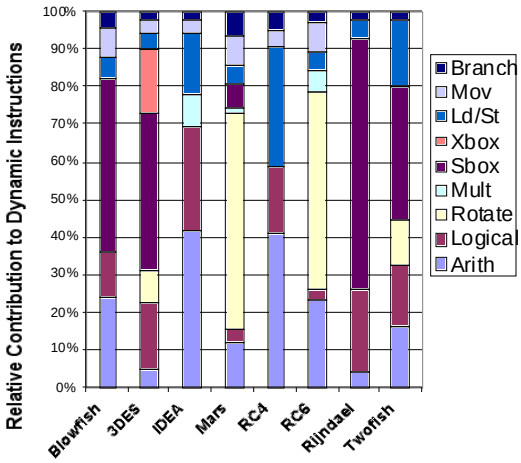

Figure 5. Breakdown of Cipher Operations.

processing limits the bandwidth requirements on this interface.

The CM processing elements block waiting on a request from the host processor. When a request is dispatched to the CM processor, it uses to the request code to initiate the correct handler function. When CM processing is complete, the tagged result of the computation is pushed into the OutQ for reception by the host processor. The host processor requests are tagged with a unique ID that can be used to identify requests as they complete and exit the OutQ. The unique ID permits requests to complete out of order as may be the case with varied processing demands on CM processors. Also contained in the CM request is a session identifier that names a unique communication channel being processed in the CM.

The CM requests specify an operation for the CM to perform on the incoming data. Operations include:

- Create a private key session. This request specifies the cryptographic algorithm, operating mode (e.g., electronic codebook *vs.* chaining mode), and the private key of the session. Algorithm setup is performed during this request, creating key-specific substitution tables.
- Delete a private key session. This request releases all storage associated with a session.
- Encrypt/Decrypt data. The requested data is processed and the resulting ciphertext or plaintext is returned in the result packet.

Additional administrative requests are supported that allow the host processor to initialize CM processor memory and redirect execution of individual CM processors.

Requests arriving for CM processing are dispatched from the InQ to CM processing elements by the request scheduler. Requests are distributed first to a free CM processor. If multiple CM processors are free, the request is dispatched to the CM processing element that most recently processed a request in the same session. If there are no free CM processors in the same session, the request is assigned to the least recently used CM processor. Directing requests to CM processors in the same session reduces the setup time to service the request, and when multiple CM processors are free but not in the same session, the least recently used CM processor is likely to contain an unneeded session context.

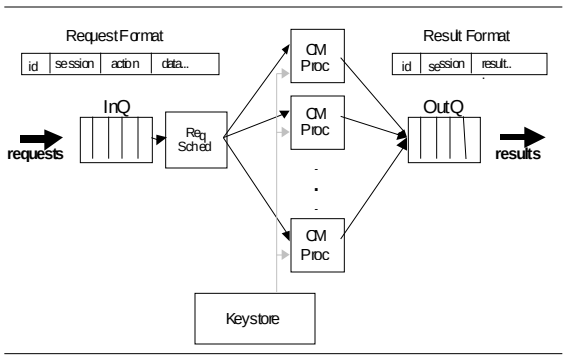

Figure 6. High-level Schematic of the CryptoManiac Architecture.

The *keystore* is a high-density storage element that contains key-specific storage such as key data and substitution tables. The keystore permits the CryptoManiac to process simultaneous sessions on the same CM processor by storing key-specific data in the shared keystore. When a new context is loaded into a CM processor, key specific data is transferred over a high performance interface to internal CM storage. This data includes substitution data, permutation counters, and other internal algorithm state, at most 5k bytes for any of the algorithms implemented. Key setup is quite expensive for many algorithms, thus performance is greatly improved by having a convenient place to store key-specific data. The keystore is an optional component to the CryptoManiac design, it is only required when multiple sessions must be serviced simultaneously. In single session applications, such as virtual private networks and secure disk processing, the keystore is not required.

3.2 Processing Element Architecture

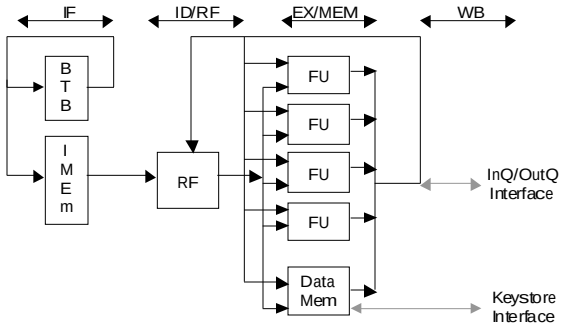

Figure 7. High-level Schematic of CryptoManiac Processing Architecture.

The CM processing element is a simple 4-wide 4-stage VLIW processor as shown in Figure 7. Each cycle the pipeline fetches a single statically scheduled VLIW instruction word that contains four independent instructions (or NOPs if independent instructions are not available).

These four instructions access the register file in parallel while decoding. In the execute stage, instructions perform up to four parallel functional unit operations. In the writeback stage of the pipeline (WB), the results of the CM instruction are written back to the register file.

The front end of the CM pipeline contains a simple branch target buffer (BTB) used to predict branch targets. The BTB contains the target address of a branch, any instruction that hits in the BTB is considered a taken branch. When a VLIW word contains multiple branches, the BTB always makes a prediction based on the last taken branch. Most branches in cipher codes are trivial to predict as nearly all branches are taken branches at the end of cipher kernel loops. Instruction memory is accessed in parallel with the BTB, returning a VLIW instruction word at the end of the processor cycle. Due to the small working set size of cipher algorithms, a very small BTB with 16 entries suffices. Moreover, the instruction memory need not be large. 1K bytes is sufficient to hold any of the cipher algorithms. If key setup codes are kept off-chip, for example, by running setup codes on the host processor, many cipher kernels could be stored in a single 1K instruction memory.

During instruction decode, instructions access the register file. To support instruction combining, the register file supports three operand reads and one write per cycle per instruction. In the EX/MEM stage, instruction operations are executed, including loads and stores. Data memory need not be large, since key tables are stored in SBOX caches within the functional units (detailed in Section 4.1); a 4K-byte data SRAM suffices for all the algorithms we implemented.

The execute stage includes four 1K-byte static RAM SBOX caches, used to speed up substitution operations. Each SBOX cache contains a 1K byte page-aligned substitution table. The alignment restriction reduces address generation to a single bit-wise concatenation of the table address and the table index. The details of this design are described in [5].

3.3 Instruction Set Architecture

Figure 8 gives a brief overview of the CryptoManiac instruction set architecture. Instructions are 32 bits in length. Each instruction contains three input registers and one output register. Three input operands are required to take advantage of the instruction-combining feature. The basic operations implemented by the combining instruction set are identical to those proposed in [5].

In a conventional microarchitecture, regardless of the latencies of instructions, each instruction takes one or more clock cycles to complete. This is the case for even very low latency instructions such as XORs and ANDs. Earlier analyses revealed a high frequency of these low latency instructions, resulting in inefficient use of the processor clock cycle. Further analyses of the cipher kernels reveal that arithmetic operations are often followed by logical operations. This property is endemic to cipher algorithms because the mixing of linear and non-linear operations prevents attacks using simple linear analysis. We can leverage this property to better utilize the processor clock cycle by combining arithmetic and logical operations within a single cycle.

bundle := <inst><inst><inst><inst>
inst := <op pair><dest><operand1><operand2><operand3>
operation pair :=
  <short><tiny>|<tiny><short>|<tiny><tiny>|<long><nop>
tiny := <xor>|<and>|<inc>|<signext>|<nop>
short := <add>|<sub>|<rot>|<sbox>|<nop>
long := <mul>|<mulmod>

Examples:

Instruction	Expression
Add-Xor R4, R1, R2, R3	R4 <- (R1+R2)⊗R3
And-Rot R4, R1, R2, R3	R4 <- (R1&R2)<<<R3
And-Xor R4, R1, R2, R3	R4 <- (R1&R2)⊗R3

Figure 8. CryptoManiac ISA in CNF form.

Our *instruction-combining* architecture divides operations into three classes: *tiny*, *short*, and *long*. Tiny operations include all logical operations and sign extension; short operations include arithmetic operations, rotates, and substitutions; multiplies are classified as long operations. Function units contain datapath networks that allow any pairs of tiny operations or short/tiny operations to execute together in a single cycle. Therefore, we combine general arithmetic instructions with logical instructions, substitutions with logical instructions, and finally rotate operations with logical instructions. Timing analyses indicate a multiplication operation can complete in under three cycles. Modular multiplication operations are implemented using one regular 16-bit multiplication followed by two 16-bit parallel additions and two levels of MUX'ing. This algorithm is derived from the Chinese remainder theorem detailed in [28]. Modular multiplication can be completed in just under three cycles.

4 Design Analysis

4.1 Methodology

The main loops of the cipher kernels are hand-optimized for the CryptoManiac instruction set. Hand-optimization of the kernels includes selection of instruction combinations and placement of instructions within VLIW instruction words. Instruction combining was performed by analyzing the known data dependencies of each kernel loop and pairing 3-input short-tiny, tiny-short, or tiny-tiny instruction combinations for minimal cycle counts. Instruction schedules are generated by analyzing kernel dependence graphs such that instructions on the critical path are executed as early as possible. We generated kernel schedules for a variety of processor widths and with/without instruction combining. We used the 4-wide instruction-combining model as our baseline model. Three more designs were evaluated in detail, including a 3-wide VLIW with combining (3WC), a 2-wide VLIW with combining (2WC), and a 4-wide VLIW without combining (4WNC). We validated our hand schedules with a super-optimizer that given a kernel dependence graph as an input can generate all possible schedules for a given architecture, keeping only

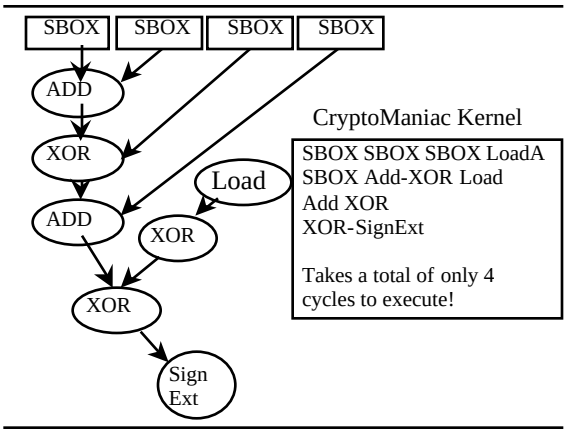

Figure 9. Blowfish Dependence Graph and Scheduled VLIW Code.

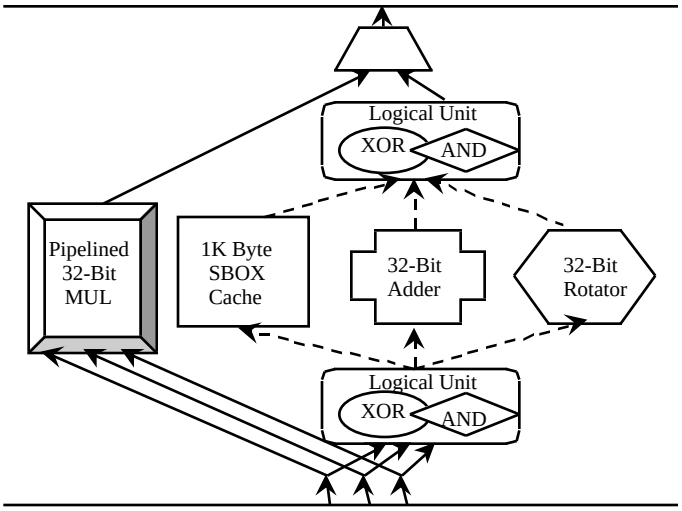

Figure 10. High-level Schematic of a Single Functional Unit.

those with the best performance and lowest resource requirements. In a few cases, we were able to improve upon the earlier hand schedules.

Table 1. Optimization of Kernel Loop Cycle Counts.

Encryption Kernel Cycle Counts (per round)					
	Alpha	4WC	3WC	2WC	4WNC
Blowfish	9.58	4	4	6	5
3DES	23.56	7	8	9	12
IDEA	91.95	14	14	17	15
Mars	28.86	9	9	9	10
RC4	11.49	8	8	8	9
RC6	23.24	7	7	7	9
Rijndael	33.84	9	11	17	10
Twofish	37.36	7	8	11	8

Table 1 lists kernel cycles per round for each original (Alpha only) and optimized cipher. The kernel cycles per round for the Alpha experiments are fractional because the code is dynamically scheduled, therefore instructions from different rounds can overlap in the same cycle. The number of cycles per round is equal to the total number of cycles to encrypt a block divided by the number of rounds. An example schedule of the Blowfish kernel for a 4-wide combining architecture is illustrated in Figure 9.

As shown in Table 1, all optimized kernels take fewer cycles than the original Alpha to execute. This result does not necessarily mean better performance because we have not yet considered the frequency at which the CryptoManiac can operate. A performance metric that combines both clock cycle time and kernel cycle counts is discussed in Section 4.3.

To gauge the cycle time of the design, a Verilog hardware model was built. The execution stage of the VLIW machine, along with full-crossbar bypass logic, and input/output queues was written in Verilog HDL. We used Synopsys logic synthesis tools [37] to evaluate design timing, area, and power. The synthesis tool accepts Verilog HDL blocks and synthesizes them according to timing constraints given. The design component library uses a 0.25um standard cell library to predict the final timing and area. To assure optimal clock speed, timing constraints were tightened in 0.25ns intervals until the synthesis failed. A 3% clock skew and interconnect wire delays were modeled as well.

Our baseline model requires four functional units to support 4-wide issue with instruction combining. As shown in Figure 10, each functional unit consists of two logical units, one adder, one 1k-byte SBOX cache, and one rotator. Only two of the four functional units contain a multiplier since none of the kernels require more than two multiplies per cycle. The logical unit can perform a XOR or AND operation as specified by the opcode of the combined instruction. Two logical units are available to provide the flexibility of logical operations at either the beginning or the end of the processor clock cycle. The rotator is implemented using a barrel shifter.

SBOX cache timing and area analysis was performed with Cacti 2 [9], a tool for estimating memory components. A 1k byte cache is roughly 0.3mm X 0.3mm, which is 0.09 mm^2 in 0.25um technology. Timing analyses indicated that the SBOX cache latency was not on the critical path of the function unit.

We explored both full and half crossbar configurations for data bypass in the execute stage. We quickly discovered that the performance impact of using a half crossbar was too great (up to 40% slower) due to many additional move instructions needed to transfer values between unconnected function units. This is not surprising given the nature of cryptographic kernels, where bit-diffusion operations require communication between all functional units.

4.2 Physical Design Characteristics

Table 2 shows the timing, area, and power consumption results for various configurations. A 4-wide CryptoManiac with instruction combining has an estimated clock cycle of 2.78ns, yielding a CryptoManiac processor that runs at 360MHz! The same configuration has an area of 1.39mm X 1.39mm (1.93mm^2), which is roughly 1/100th the size of a 600MHz Alpha 21264 processor (200mm^2) in the same technology. The average estimated power consumption of the 4-wide combining model running at 360MHz with a V_{dd}

Timing and Area Estimates for Various CryptoManiac Design Configurations				
	4W Combining	3W Combining	2W Combining	4W Non-Comb
Timing Result	2.78 ns	2.66 ns	2.54 ns	2.76 ns
Area Result	1.39mm X 1.39mm	1.33mm X 1.33mm	1.26mm X 1.26mm	1.3mm X 1.3mm
Power Result	606.37 mW	593.51 mW	568.50 mW	586.86 mW
Synthesis Constraint	3 ns	3 ns	3 ns	3 ns
Critical Path(s)	bypass-logic-add-logic	bypass-logic-add-logic	bypass-logic-add-logic	adder

Table 2. Timing and Area Results for CryptoManiac.

of 2.1V is 606mW. The Alpha processor running at 600MHz dissipates 75W. The power consumption results, although an estimate, seem reasonable because the relative power with CryptoManiac running at 600MHz is 1.01W ($1/75^{th}$ of the Alpha 21264), which is proportional to the area differences.

The chip area of the CryptoManiac co-processor is likely over-estimated. Synthesized designs tend to be larger than their hand-optimized counter parts, and the Cacti 2 area estimates are known to be larger than physical designs we have constructed in the past. An experienced design team could likely produce a smaller and faster design.

4.3 Performance Analysis

In Figure 11, we show the performance of the four models studied, plus the original Alpha, and two versions of the ISA extensions studied previously [5]. A performance metric of megabytes encrypted per second is used to show the encryption performance of each algorithm. The ISA+ model is a 600MHz Alpha 21264-like processor with cryptographic instruction set enhancements. The model is implemented using the SimpleScalar tool set, details can be found in [5]. The ISA++ model has the same micro-architecture as the ISA+ model plus four 1k-byte SBOX caches to optimize substitution performance. In the 4-wide combining (4WC) model, there are four functional units, two multipliers, and support for instruction combining. All other configurations are derived from this model. The 3WC and 2WC models reduce the number of functional units to three and two, respectively. These configurations also have two multipliers each. The 4WNC model has four functional units and two multipliers, and it does not support instruction combining. In this design, an XOR instruction would take one cycle to execute, as would an Add instruction. This design benefits from a more efficient register file, since each instruction has only two input operands.

The encryption rates for the four models we designed are measured using 4K byte sessions with 128 byte blocks. Encryption rates are calculated in MB/s by dividing cycles/byte into the clock period of the hardware model. As shown in Figure 11, we were able to achieve an encryption rate as high as 64 MB/s for Rijndael (the new AES standard) which is 2.25 times faster than the 600MHz Alpha 21264 workstation. All kernels except RC4 gained in performance, ranging from 32% to 290% better than the 600MHz Alpha. RC4 is the only kernel that performs worse than the baseline Alpha processor due to ample aliasing effects described in Section 2. RC4 writes into its key table, creating many ambiguous memory dependencies that lead to poor schedules on the VLIW architecture. Processors that are dynamically scheduled can run RC4 much faster. Nevertheless, the 4-wide combining CryptoManiac configuration ran, on average, 1.2 times faster than the Alpha processor. The Alpha processor with ISA extensions faired much better, out-performing the 4WC CryptoManiac in a few experiments. Keep in mind that the CryptoManiac design is much more cost-effective than a conventional out-of-order processor.

The dashed lines in Figure 11 represents various real world performance targets. A T-3 line can be saturated at a speed of 5.375 MB/s. An MPEG-4 HDTV transmits at 8MB/s. An OC-3 line has the bandwidth of 19.375 MB/s and an OC-12 line has the bandwidth of 77.5 MB/s. All of the kernels running on a CryptoManiac can saturate a T-3 line or an MPEG-4 HDTV line. All but two kernels met the bandwidth requirement for an OC-3 line.

Figure 12 illustrates the tradeoff between performance and area for variety of physical designs. We examined designs from two to eight instructions wide, with and without instruction combining, for the Rijndael and 3DES ciphers. Parallelism exhibited by each algorithm has a direct effect on the encryption rate. MARS, RC4, and RC6 benefited little from additional issue bandwidth. These kernels do, however, run faster with instruction combining. Their performance decreased significantly on the 4-wide non-combining configuration. Rijndael benefits from additional issue bandwidth; an 8-wide configuration is nearly 30% faster than the 4-wide design. Instruction combining appears to be a beneficial feature, configurations with this capability are more than 10% faster with even smaller increases in area.

4.4 System analysis

There are many potential applications of the CryptoManiac processor. In this section, we examine the performance of the CryptoManiac processor for two applications: secure web server and disk controller. We analyze the performance of general purpose and cryptographic processors using I/O trace-based simulation, measuring the response time for each processor configuration to service requests.

The secure web server experiments were driven by a network traffic trace of the WorldCup 98 official web server (www.worldcup.com), captured during a one-hour period of extremely high traffic [19]. During this one hour period, there was an average of 1971 requests per second with a total transfer rate of 8.75 MB/sec. Thirty web servers were used to service this traffic, we have assembled all the requests into a single trace for this experiment. In addition,

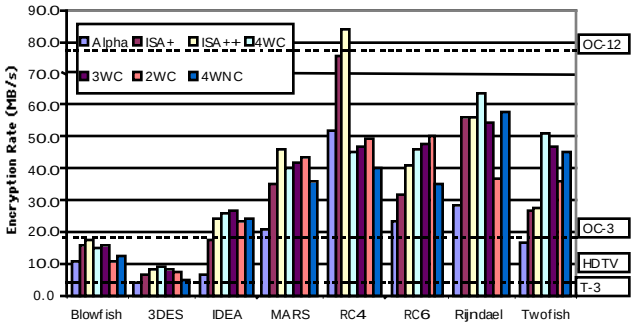

Figure 11. CryptoManiac Encryption Performance.

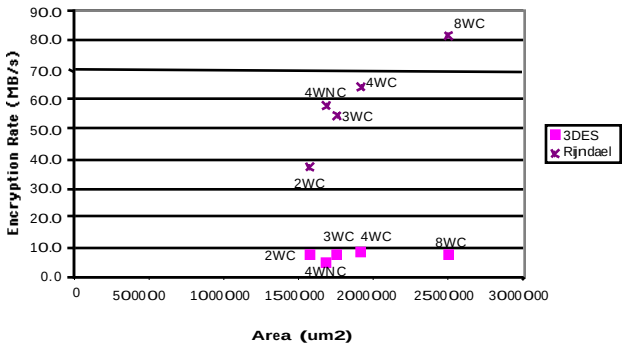

Figure 12. Five Design Models and Their Performance/Area Tradeoff for 3DES and Rijndael.

this traffic was not secure, so we've transformed it into secure requests by bracketing sessions with an SSL public key authentication [38].

In the network traffic experiments, packets are encrypted and decrypted using Chaining Block Cipher (CBC) mode, as specified by the IPSEC protocol standard [3]. In this mode of operation, the cipher text of the previous encrypted block (128 bits for Rijndael, 64 bits for 3DES) is XOR'ed with the plaintext of the next block before it is encrypted. Chaining blocks increases the strength of cipher algorithms by reducing correlation between the plaintext and ciphertext, at the expense of parallelism. Packet sizes are limited to 1500 bytes, as specified by the IPSEC protocol standard.

The secure disk experiments are driven with a trace of accesses to a 9-disk array of 9.1 GB Quantum Atlas 10K disks [31]. The trace was taken from the DiskSIM disk simulator trace library [14]. During the trace, the disk is heavily loaded, with an average transfer rate of 16.7 MB/sec. The accesses to all the disks are combined into a single trace for the purpose of our analyses. Disk blocks are encrypted using CBC mode encryption in 512 byte encryption units (the minimum disk transfer size).

We examine the performance for single and multiple processor configurations. With multiple processors, network packets can be processed in parallel if they are from different sessions (i.e., different connections from different IP addresses). Within a session, cipher block chaining requires that stream packets be processed serially. For the disk experiments, sector data is processed in chaining block mode, but different disk sector accesses (a sector is 512 bytes) may be processed in parallel with multiple CryptoManiac processors.

For all experiments, we only consider the performance of cryptographic processing, all other processing tasks such as OS, web server, and database operations are assumed to be offloaded to other processor components. We assume that public key authentication (used once at the beginning of each session from a unique IP address) is implemented with two dedicated public key processors. Each public key authentication takes 3.2msec, this timing is based on the performance of the HiFn 6500 public key processor [17].

Once authentication completes, the private keys and key tables are stored in the keystore for the entire length of the user session. When a session context is loaded into a processor, for example to process the first packet or to change the session context of a CryptoManiac processor, we assume that the loading of context data takes 720ns. This timing is based on a keystore constructed from a 400 MHz RDRAM [27], which can access key table data in two 40ns accesses (across the rows of 32 banks), plus a 1k byte bus transfer at 16-bits per 10ns plus bus overheads.

Figure 13 graphs the response time of the Rijndael (right) and 3DES (left) ciphers for the secure web server (top) and disk controller (bottom) application. For the 3DES experiments we also show the simulator performance of the HiFn 7751 encryption processor [17]. The 7751 is a high-end encryption processor used in VPN routers and other high bandwidth secure communication applications. The 7751 includes a hardware implementation of the 3DES algorithm capable of encrypting data at 10.375 MB/sec (in IPSEC-compatible CBC mode).

As shown in Figure 13, network traffic processing requires more resources than disk I/O processing. This is due to the fact that network I/O processing *a)* has less parallelism due to the chaining of packets within connections, and *b)* switches contexts often necessitating the extra delay of loading key-specific data from the keystore. The disk I/O workload, even though at higher sustained bandwidth, operates within a single context (and thus does not access the keystore). The disk workload also has ample parallelism, since different disk blocks (512 bytes in size) can be processed in parallel on different CM processors.

To keep cryptographic processing overheads for network traffic low, say below 5% for a short 40msec transfer delay, additional transmission delays due to cryptographic processing must be no more than 2msec total, or no more than 1msec (1000usec) on each end of a network transfer. For the 3DES network I/O experiments, an acceptable level of overhead requires at least three CM processor or three 7751 processors. For the Alpha processor experiments, acceptable network delays (with 3DES) require six Alpha 21264 processors or four Alpha 21264 processors with cryptographic ISA extensions (labeled Alpha+). With Rijndael, performance is much better; two processors suffice for all the experiments.

Disk transfers for the Atlas 10K drive average 16msec in length, as a result, overheads can be limited to 5% if the increase in sector transfer latency is no more than 800ns. The disk I/O experiments cannot meet this goal for any 3DES configuration examined due to disk block processing

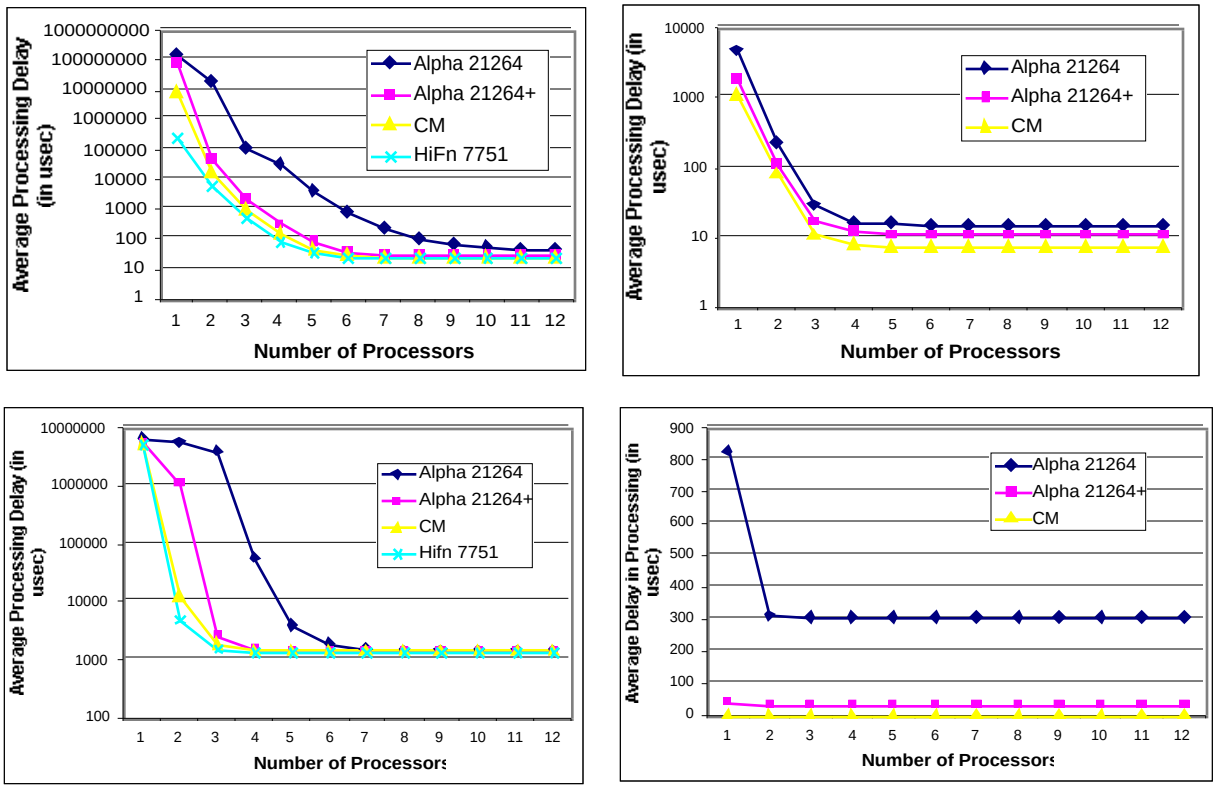

Figure 13. Average Delay in Processing (usec) vs. Number of Processors. Top two graphs are Network traces, and the bottom two graphs are Disk I/O traces. Left two graphs are for 3DES and right two graphs are for Rijndael.

delays. Minimum processing latency was always greater than 800ns. For Rijndael, performance is again much better with all configurations able to service disk I/O with acceptable delay using only a single processor.

5 Related Work

Many algorithm-specific hardware implementations have been described in the literature. IBM's original DES proposal described a hardware implementation [13]. Shiva [36], IBM [20], Chrysalis-ITS [8], and Hi/FN [17] all offer high speed hardware implementations of the DES and 3DES algorithms. Hardware implementations have also been described for IDEA [23], Twofish [23], and Blowfish [23]. The FPGA research community has also shown that public key cipher algorithm performance can be improved using FPGA-based implementations [26].

We are aware of only two previous proposals to add instruction set support for private key symmetric cryptography. Our previous proposal [5] examined extensions to the Alpha instruction set to improve private key processing performance. Shi and Lee proposed adding an instruction (GRP) that supports efficient software implementations of general bit permutations [35].

6 Conclusions and Future Work

The growth of the Internet as the primary vehicle for secure communication and electronic commerce has made efficient cryptographic processing a key factor of good system performance. In this paper, we demonstrated that a hardware-software co-design provides excellent performance while maintaining the flexibility to support new algorithms in the field. To motivate our design, we analyzed the characteristics of eight symmetric-key cipher kernels. We showed that they lack branch or memory bottlenecks, have few unknown dependencies, and offer little headroom for performance improvement on traditional architectures.

We presented the CryptoManiac co-processor, a 4-wide VLIW processor with no cache and a simple branch predictor, built around a clock cycle that combines arithmetic and logical operations. The execution stage of the processor contains four functional units each equipped with an adder, a 1k-byte SBOX cache for efficient substitutions, two logical units to support instruction combining, and a rotator. There are also two multipliers per processing element, extended for excellent performance when executing modular multiplies. The instruction combining feature not only made executing instructions with varied latencies efficient, but also demonstrated how our design catered to cryptographic processing.

We evaluated different design configurations by building detailed hardware models of varied widths and capabilities. We then calculate encryption rate by synthesizing the models to obtain timing estimates. Our systematic approach allowed us to study the tradeoffs between chip area and performance. We showed that the highest-performing and most cost-efficient design is the 4-wide combining configuration. Rijndael, the new AES standard, runs 2.25 times faster on a 360MHz CryptoManiac. Our analysis of the original and optimized algorithms suggests that there is more opportunity to speed up cryptographic processing. We are considering improved functional unit designs as well as more aggressive circuit implementations.

Our results make a very strong case for the deployment of cryptographic co-processors, however, we believe the results in this paper have stronger implications for the computer architecture community as a whole. With an additional 1% area (for an Alpha 21264 design), we were able to affect a 20% performance improvement over a broad class of cipher algorithms, with individual algorithms benefiting as much as 190%. This is a striking result considering that many commercial design teams use a rule of thumb that any optimization that returns 1% performance improvement for 1% area is a good one. This result is further underscored by the fact that our design is completely synthesized, if the talents of an experienced design team were marshaled to this task, the resulting design would be smaller, faster and cooler.

The reason for these striking results is simple - an *application specific processor* design can achieve a level of efficiency that is impossible for general purpose designs to attain. Our application specific design contains none of the baggage necessary to execute non-cryptographic workloads, making the resulting design smaller and cooler. In addition, our limited application domain creates opportunities to optimize the implementation, yielding superior performance results. Going forward, we are working to assess the cost of programmability in the CryptoManiac. A dedicated Rijndael implementation is under development that will be compared to the design presented in this paper. In addition, we are developing application specific processors for other application domains. Through this work we hope to demonstrate that application specific optimization can be a powerful tool for improving system performance and cost.

References

[1] *Advanced EncryptionStandard (AES) Development Effort.* US Government, http://csrc.nist.gov/encryption/aes/.

[2] M. Arlitt and C. Williamson. Web server workload characterization: The search for invariants. *Proceedings of the ACM SIGMETRICS '96 Conference*, April 1996.

[3] R. Atkinson. Security architecture for the internet protocol. *IETF Draft Architecture ipsec-arch-sec00*, 1996.

[4] T. Blum and C.Paar. Montgomery modular exponentiation on reconfigurable hardware. *Proceedings, 14th IEEE Symposium on Computer Arithmetic*, pages 70-77, 1999.

[5] J. Burke, J. McDonald, and T. Austin. Architectural Support for Fast Symmetric-Key Cryptography. *Proceedings of ASPLOS*, 2000.

[6] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-97-1342, Unniversity of Wisconsin, Madison, June 1997.

[7] C. Burnwick and et al. *The Mars Encryption Algorithm.* IBM, http://csrc.nist.gov/encryption/aes/round2/AESAlgs/MARS, 1999

[8] *Chrysalis-ITS Corporation.* http://www.chrysalis-its.com

[9] *Compaq Corporation.* http://www.research.compaq.com/wrl/techreports/abstracts/93.5.html.

[10] *Counterpane Systems.* http://www.couterpane.com

[11] *CryptSoft Technologies.* http://www.cryptsoft.com, 2000.

[12] J. Daemen and V. Rijmen. *AES Proposal: Rijndael.* http://csrc.nist.gov/encryption/aes/round2/AESAlgs/Rijndael, 1999.

[13] D. W. Davis and W. L. Price. *Security for Computer Networks*. Wiley, 1989.

[14] DiskSIM simulator. http://www.ece.cmu.edu/~ganger/disksim.

[15] P. Fergguson and G. Huston. What is a VPN. *http://www.employees.org/ferguson/vpn.pdf*, 1998.

[16] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Palo Alto, Calif.: Morgan Kaufmann, 1990.

[17] Hi/Fn Corporation. http://www.hifn.com

[18] J.-H. Hong and C.-W. Wu. Radix-4 modular multiplication and exponentiation algorithms for the RSA public-key cryptosystem. *Design Automation Conference (ASP-DAC 2000)*, pages 565-570, 2000.

[19] *Internet Traffic Archive.* http://ita.ee.lbl.gov.

[20] *S/390 and OS/390 Crryptography.* htpp://www.s390.ibm.com/security/cryptography.html.

[21] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World.* Prentice Hall PTR, 1995.

[22] J. Keller. A superscalar alpha processor with out-of-order exeecution. *9th Annual Microprocessor Forum*, 1996.

[23] Xuejia Laai. *On the Design and Security of Block Ciphers.* Hartung-Gorre Veerlag, 1992.

[24] M. H. Lipasti and J. P. Shen. Exceeding the dataflow limit via value prediction. In *29th International Symposium on Microarchitecture*, December 1996.

[25] M. S. Merkow, CCP, and J. Breithaupt. *The Complete Guide to Internet Security.* AMACOM, 2000.

[26] U. Meyer-Base and R. Watzel. A comparison of DES and LFSR based FPGA implementable cryptography algorithms. *3rd International Symposium on Communication Theory and Applications*, pages 290-298, 1995.

[27] *Micron Corporation*. http://www.micro.com/rdram.

[28] P. L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44(170):519-521, April 1985.

[29] Stephen Mooore. *Enhancing Security Performance Through IA-64 Architecture.* Intel Corporation, http://developer.intel.com/design/security/rsa2000/itanium.pdf, 1999.

[30] *An Introduction to Cryptography*. Network Associates, Inc., http://www.pgpi.org/doc/pgpintro/, 1999.

[31] *Quantum Corporation*. http://www.quantum.com.

[32] R. L. Rivest and et al. *The RC6 Block Cipher*. RSA Security, http://csrc.nist.gov/encryption/aes/round2/AESAlgs/RC6.

[33] *RSA Security.* http://www.rsa.ocm.

[34] B. Schneier and et al. *Twofish: A 128-Bit Block Cipher*. Couterpane Systems, http://csrc.nist.gov/encryption/aes/round2/AESAlgs/Twofish, 1998.

[35] Z. Shi and R. B. Lee. Bit permutation instructions for accelerating software cryptography. *Proc. Of the IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 138-148, 2000.

[36] *Shiva Corporation*. http://www.shiva.com.

[37] *Synopsys.* http://www.synopsys.com.

[38] *The SSL Protocol, version 3.0*. Netscape, Inc., http://home.netscape.com/eng/ssl3/draft302.txt, 1999.