Fast Parallel Table Lookups to Accelerate Symmetric-Key Cryptography

A. Murat Fiskiran and Ruby B. Lee
Department of Electrical Engineering
Princeton University
{fiskiran, rblee}@princeton.edu

Abstract

Table lookups are one of the most frequently-used operations in symmetric-key ciphers. Particularly in the newer algorithms such as the Advanced Encryption Standard *(AES), we frequently find that the greatest fraction of the execution time is spent during table lookups, varying between 34% and 72% for the five representative ciphers we consider: AES, Blowfish, Twofish, MARS, and RC4. In order to accelerate and parallelize these table lookups, we describe a new* parallel table lookup (**ptlu**) *instruction. Our synthesis results indicate that such an instruction can be added to a basic RISC processor with no cycle time impact. We compare the performance of the* **ptlu** *instruction with the speedups available through more conventional architectural techniques such as multiple-issue execution. We find that the performance benefits of using the* **ptlu** *instruction can be far higher than increasing the number of instructions executed per cycle in superscalar or VLIW processors.*

1. Introduction

Symmetric-key cryptography is used to provide data confidentiality on public communication networks such as the Internet [14][18]. This is achieved by encrypting a *plaintext* message *P* using a symmetric-key algorithm (cipher) and a secret key *K*. The encrypted data (*ciphertext*) is then transmitted to the receiver, where it can be decrypted using the same cipher and secret key. Examples of widely-used symmetric-key ciphers are the *Data Encryption Standard* (DES), *Triple-DES* (3DES), and the *Advanced Encryption Standard* (AES) [1][18].

Symmetric-key ciphers usually have an *iterated round* structure, where a short sequence of operations (called a *round*) is repeated multiple times on the plaintext block to compute the ciphertext. The operations in a round are chosen to rapidly achieve cryptographic *confusion* and *diffusion*, which obscure the relationship between the plaintext and the ciphertext [18]. Commonly-used round operations include table lookups, modular addition and subtraction, logical operations, shifts and rotates, multiplication, and bit permutations. Table lookups in particular are used very frequently by the newer symmetric-key ciphers such as AES. Of the five finalist ciphers in the AES Development Effort [2], four used table lookups in their round structure.

Our previous studies [8][9] showed that the frequent use of table lookups in ciphers causes the execution time to be dominated by the overhead instructions required for effective address computations. For example, table lookups may account for as much as 72% of the aggregate AES execution time when it is implemented using a typical RISC-like instruction set like MIPS32 [15].

Our first contribution in this paper is the execution time analysis of five symmetric-key ciphers that use table lookups in their round structure (Section 2). We then describe how to accelerate and parallelize these table lookups by using a new **ptlu** instruction (Section 3). In Section 4, we compare the performance benefits of our proposal with those of other architectural techniques such as superscalar execution and *wordsize scaling* [11][12]. Section 5 reviews the related past work, and Section 6 is the conclusion.

2. Cipher suite

The symmetric-key ciphers we consider in this study are shown in Table 1. AES is the NIST standard for block encryption and it is included in many widely-used security protocols such as IPSec, TLS, and SSH [1]. Due to its importance, we consider AES with 128, 192, and 256-bit keys, denoted AES-128, AES-192, and AES-256 respectively. For the remaining ciphers, we only consider the typical key size. Blowfish is a symmetric-key cipher designed by Schneier [18] and it is used in many popular applications such as GPG,

This work was supported in part by NSF ITR CCR-0326372.

To be published in the *Proceedings of the International Conference on Information Technology Coding and Computing (ITCC), Embedded Cryptographic Systems Track,* 4-6 April 2005, Las Vegas, Nevada, USA.

Table 1: Cipher suite

Cipher	Block Size (bits)	Key Size (bits)
AES	128	128, 192, 256
Blowfish	64	128 bits typical
MARS	128	
Twofish	128	
RC4	8	

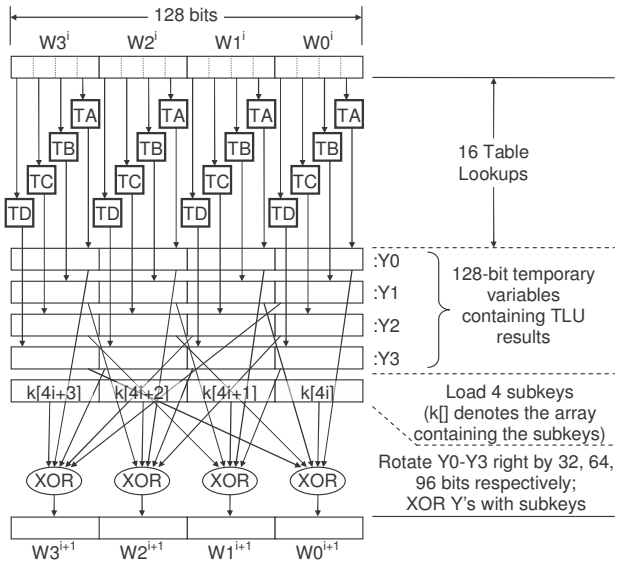

Figure 1: AES round

Table 2: Base instruction set

Class	Instructions
ALU	add, addi, sub, subi, and, andi, or, ori, xor, xori, not, loadi
Shift	sra, srai, srl, srli, sll, slli, rotr
Multiply	mul.lo, mul.hi
Memory	load, store (base+displacement addressing)
Branch	beq, bne, bg, bge, call, return, trap

Table 3: Breakdown of execution time

	% Execution Time				
	AES	Blowfish	MARS	Twofish	RC4
Table lookup	72	36	34	43	54
Arithmetic	-	26	10	15	14
Logical	24	34	18	32	26
Multiplication	-	-	19	-	-
Fixed shift/rotate	-	-	5	4	-
Variable rotate	-	-	8	-	-
Other	4	4	6	6	6
Total	100	100	100	100	100

Table 4: Structure of lookup tables

Cipher	Num. Tables	Num. Entries	Entry Size (bits)	Lookups per Block of Encryption
AES	4	256	32	160 (AES-128), 192 (AES-192), 224 (AES-256)
Blowfish	4	256	32	64
MARS	2	256	32	80
Twofish	4	256	32	128
RC4	1	256	8	3 reads, 2 writes

SSH, and JAVA. MARS and Twofish were two of the five finalist ciphers in the AES Development Effort [4][19]. RC4 is a stream cipher used in the wireless LAN standard IEEE 802.11 [10][18].

2.1. Table lookups in AES

To illustrate how table lookups are typically used in symmetric-key ciphers, we show the detailed round structure of AES in Figure 1. The input to the i^{th} AES round is a 128-bit block made up of four 32-bit words (labeled $W3^i$-$W0^i$ in Figure 1). There are four 1 kB tables, labeled TA-TD. Each table has 256 entries, where each entry contains 4 bytes. During the round, the rightmost byte of each word is used as an index into TA; the next byte is used as an index into TB; and so on, until all tables are accessed four times. Finally, the four table lookup results (for each input word) and a round subkey are XORed as shown.

Of the remaining ciphers in our suite, Blowfish, MARS, and Twofish are similar to AES in that they use multiple 32-bit-wide tables with 256 entries. In contrast, RC4 uses a single 256-entry table, where each entry is byte-wide.

2.2. Execution time analysis

We use the PLX toolset [12][17] to generate execution profiles for each cipher. For baseline performance data, we implement the ciphers in assembly using a RISC-like instruction set, which is shown in Table 2. We call this the *Base ISA*. The simulator is configured to model a 32-bit in-order single-issue processor with single-cycle instructions (except for multiplication which has 3-cycle latency). We also assume a perfect memory model with single-cycle load/store instructions. Our results are summarized in Table 3, which shows the main operations used by each cipher and the fraction of the execution time consumed by these. Table 4 summarizes the structure of the lookup tables used by the ciphers. We show the number of tables, number of entries per table, entry size, and total number of lookups used per block of encryption. Based on Table 3 and Table 4, we make the following observations:

- All these ciphers spend the largest fraction of their execution time during table lookups.
- Tables are few (at most four) and have constant size. Except for RC4, all table accesses are reads.
- Number of entries per table is small (256 for all ciphers), and the data read is either 8 or 32 bits.
- The round structures generally permit the table lookups to be parallelized. For example, all 16 lookups in an AES round (Figure 1) can be performed in parallel, constrained only by hardware resources.

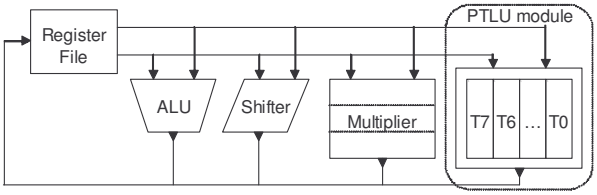

Figure 2: Single-issue processor with ptlu

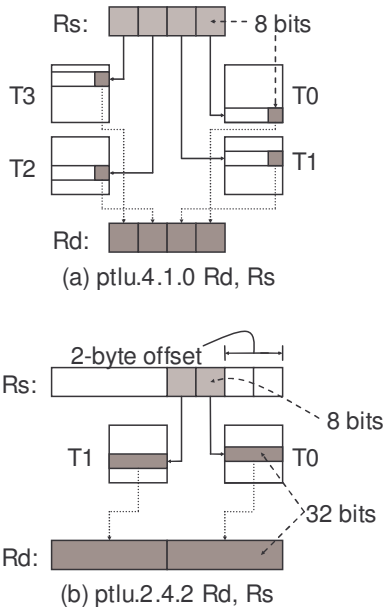

Figure 3: Examples of ptlu

Hardware support can be used to exploit some of these properties to parallelize and accelerate the table lookups. In the next section, we describe how a new *parallel table lookup* instruction can be used to do this.

3. Hardware support for fast table lookups

3.1. ptlu instruction

We propose extending the base ISA in Table 2 with a new instruction called **ptlu** (for *parallel table lookup*). Figure 2 shows the modified datapath of a single-issue RISC processor that implements this instruction. In addition to the standard functional units ALU, shifter, and the 3-stage pipelined multiplier, there are eight small on-chip tables, labeled T0-T7. Each table has 256 entries, where each entry is at most as wide as the processor wordsize. To limit cost and power in co-designed embedded systems, it is possible to implement fewer than eight tables, or use tables that are narrower than the processor wordsize. Assuming all eight tables are implemented, the total size will be 8 kB with 32-bit-wide tables, 16 kB with 64-bit-wide tables, and 32 kB with 128-bit-wide tables.

The **ptlu** instruction used to access these tables has the following format with three sub-op fields:

ptlu.*lookup.data.offset* Rd, Rs

The 3-bit *lookup* field specifies the number of table lookups that will be performed in parallel. Because each table has a single read port, at most eight lookups can be performed in parallel on a processor with eight tables. If fewer than eight tables are implemented, then the number of maximum parallel lookups will be constrained by the number of tables.

The 3-bit *data* field specifies the size of the data read from the table(s), as 1, 2, 4, 8, or 16 bytes. For example, when *data* = 1, the rightmost byte of the selected table entry is read and when *data* = 2, the rightmost two bytes are read. In order to accommodate the result in the destination register *Rd*, the number of parallel lookups multiplied by the data size must be equal to or less than the processor wordsize. This can be checked by the compiler statically, or by hardware at runtime. Since *data* can specify up to eight different quantities, the instruction is extensible to read tables that are wider than 16 bytes. However, such wide tables were not needed for the ciphers we have studied, including many others not presented in this paper.

The 4-bit *offset* field specifies the location of the first byte in *Rs* that will be used as an index. When multiple parallel lookups are performed, the bytes adjacent and to the left of the first index are used as the next indices.

Figure 3 contains two examples of the **ptlu** instruction. Figure 3a shows a *byte substitution* operation on a 32-bit processor that implements 32-bit tables. Four parallel lookups (*lookup* field = 4) are performed to replace each byte of *Rs* with another byte. Such lookups are used, for example, in AES key expansion [1]. The *data* field (=1) specifies that byte-sized blocks are read from the tables. The *offset* field (=0) specifies that the rightmost byte of *Rs* (byte 0) is the first index (into T0), and the next three bytes are used as indices into tables T1, T2, T3 respectively. While we only show the four tables accessed by the **ptlu** instruction in Figure 3a, the processor may have up to eight tables. In Figure 3b, we show how **ptlu** can be used to perform two parallel 32-bit lookups on a 64-bit processor that implements 32-bit tables. This type of lookup is common in AES as shown in Figure 1.

The writing of tables T0-T7 is done with the **ptlw** instruction, which has the following format:

ptlw.*table.data.offset* Rs1, Rs2

This instruction can only write a single table, selected by the 3-bit *table* field, rather than multiple tables in parallel. This limitation does not degrade encryption performance since the ciphers do not

```
# Single AES round - 128-bit processor
# R10 is the 128-bit round input
# R5  is pointer to round key array

ptlu.4.4.0   R11,     R10     # 4 TLUs on W0
ptlu.4.4.4   R12,     R10     # 4 TLUs on W1
ptlu.4.4.8   R13,     R10     # 4 TLUs on W2
ptlu.4.4.12  R14,     R10     # 4 TLUs on W3

load.16    R15, R5,  16       # load 4 subkeys

rotr    R12, R12,  32     # rotate right
rotr    R13, R13,  64     # rotate right
rotr    R14, R14,  96     # rotate right

xor     R11, R11, R12     # xor all
xor     R13, R13, R14
xor     R11, R11, R13
xor     R10, R11, R15     # R10 is new state
```

Figure 4: Optimized AES round with ptlu (128-bit processor)

require parallel table writes. Furthermore, setting up the tables is only done at initialization time.

Similar to **ptlu**, the 3-bit *data* field specifies the size of the data to be written to the selected table. The data is read from the rightmost byte(s) of *Rs2*. If the tables are wider than the data size, then zeros are written on the left to complete the table entry. Finally, the 4-bit *offset* field selects a byte from *Rs1* to be used as index into the selected table.

3.2. Optimized AES round

An optimized version of the AES round (Figure 1) using **ptlu** instructions is shown in Figure 4 for a 128-bit processor that implements 32-bit tables. Assuming that the four AES tables (TA-TD) are mapped to the four physical tables T0-T3 respectively, four **ptlu** instructions are sufficient to complete all 16 lookups in an AES round. The **load.16** instruction loads a whole 16-byte word into R15. The total number of instructions used per round is only 12. Without **ptlu**, the same round requires at least 84 instructions, most of which are overhead instructions used for effective address computations.

Another advantage of using **ptlu** is the elimination of cache read misses. Unlike data memory, where a **load** instruction can take either a single cycle (cache hit) or many cycles (cache miss), a **ptlu** access always takes a single cycle. This eliminates the variability in encryption or decryption latency.

3.3. Area and delay of lookup tables

For cost analysis of the lookup tables, we use CACTI 3.2, which is a tool for estimating the access time, area, and aspect ratio of memory components [5]. In a 90 nm process technology, we estimate a single 256-entry 32-bit-wide table to have an area of roughly 0.2 mm × 0.2 mm = 0.04 mm^2, which corresponds to about 17K minimum-sized two-input NAND gates. The total area of the eight **ptlu** tables will range from 0.32 mm^2 if the entry size is 32 bits, to 1.04 mm^2 if the entry size is 128 bits. Even with eight 128-bit-wide tables, the total size is *at most* that required by 32 kB of SRAM, which is smaller than today's L1 caches.

The access time of the **ptlu** tables is not likely to impact the processor cycle time because each table is at most 1 to 4 kB in size, which is small and hence can be very fast. To verify this, we use Synopsys tools to perform gate-level synthesis of the functional units of the processor shown in Figure 2, which implements the instruction set in Table 2. In Table 5, we compare the ALU latency to the table access time obtained from CACTI for 32-bit, 64-bit, and 128-bit processors. We assume the cycle time is determined by the ALU, and hence we do not show the latency of the shifter and the pipelined multiplier. The table access time relative to the ALU decreases as the ALU width increases. Since the table access time is always less than the single-cycle ALU latency, the **ptlu/ptlw** instructions will also have single-cycle latency.

Table 5: ALU and table lookup latency

	32- bit		64-bit		128-bit	
	ALU	TLU	ALU	TLU	ALU	TLU
Delay (ns)	0.43	0.37	0.49	0.40	0.56	0.41
Delay (normalized to *n*-bit ALU)	1.00	0.86	1.00	0.82	1.00	0.73

4. Performance

We evaluate the performance benefits of the new **ptlu** instruction compared to more traditional architectural techniques, such as *multiple-issue* execution (e.g. superscalar) and newer technology like *wordsize scaling* [11][12]. We then compare these results to the speedups obtained when **ptlu** is added to the base instruction set.

4.1. Superscalar execution

Table 6 summarizes the speedups for the ciphers when a superscalar processor is used. The second column in the table shows the cycles required per block of encryption on a single-issue basic RISC processor (with the instruction set shown in Table 2). The remaining columns show the speedups when the issue width and the number of memory ports are increased. In the notation n_1/n_2, n_1 denotes the issue width and n_2 denotes the number of memory ports.

We see that 2-way and 4-way superscalar execution with a single memory port provides significant performance improvements for all ciphers (up to

1.87×). Further increasing the issue width to 8 provides only minor additional performance (up to 2.02×). We also observe that due to the memory-intensive round structures of the ciphers, the addition of a secondary memory port increases performance significantly at any issue width.

Table 6: Speedups with superscalar execution

		Issue width / Num. Memory Ports						
Cipher	**Cycles**	**1/1**	**2/1**	**2/2**	**4/1**	**4/2**	**8/1**	**8/2**
AES-128	870	1.00×	1.58×	1.71×	1.85×	2.23×	2.02×	2.49×
AES-192	1056	1.00	1.57	1.70	1.83	2.20	1.99	2.49
AES-256	1272	1.00	1.59	1.73	1.87	2.24	2.02	2.51
Blowfish	408	1.00	1.32	1.56	1.67	1.99	1.79	2.14
MARS	747	1.00	1.36	1.63	1.68	2.08	1.84	2.22
Twofish	1538	1.00	1.55	1.77	1.87	2.19	1.97	2.51
RC4	18	1.00	1.41	1.58	1.71	2.05	1.90	2.31

Table 7: Speedups with ptlu **and wordsize scaling**

	32-bit		64-bit		128-bit	
Cipher	**w/o ptlu**	**w/ ptlu**	**w/o ptlu**	**w/ ptlu**	**w/o ptlu**	**w/ ptlu**
AES-128	1.00×	2.28×	1.00×	2.85×	1.00×	6.09×
AES-192	1.00	2.31	1.00	2.85	1.00	6.11
AES-256	1.00	2.32	1.00	2.87	1.00	6.15
Blowfish	1.00	1.88	1.00	2.04	1.00	2.24
MARS	1.00	1.34	1.00	1.61	1.00	2.18
Twofish	1.00	1.61	1.00	2.12	1.00	2.49
RC4	1.00	1.92	1.00	1.98	1.00	2.12

4.2. Subword parallelism, *ptlu*, and wordsize scaling

Subword parallelism, which is frequently used in multimedia-oriented ISAs [13], involves partitioning a processor's datapath into *subwords*, which are units smaller than a word. Multiple subwords packed in a word can then be processed simultaneously using *parallel instructions*. For example, a 64-bit processor can add four pairs of 16-bit subwords packed in two source registers using a single *parallel add* (**padd**) instruction.

Wordsize scaling, which was first introduced in the PLX multimedia ISA [11][12], allows an instruction set to be synthesized into processors with different wordsizes. By increasing the wordsize, more subwords can be packed in a word and processed in parallel without increasing the number of instructions executed. This provides higher speedups than multiple-issue techniques such as superscalar execution, and has lower implementation complexity [6][9][11][12].

To evaluate the performance impact of wordsize scaling on symmetric-key cryptography, we simulate our ciphers on 64-bit and 128-bit processors that support parallel operations on 32-bit subwords. This involves extending the base ISA in Table 2 with the following instructions: **padd** (parallel add), **psub**, **psrli** (parallel shift right logical immediate), **pslli**, **protr** (parallel rotate right). The **padd** instruction for example adds two pairs of 32-bit subwords on the 64-bit processor, and four pairs on the 128-bit processor.

Our results are summarized in Table 7 for processors with and without the **ptlu** instruction. On a 32-bit processor, the inclusion of **ptlu** provides very significant speedups for all ciphers. Here, AES shows the highest speedup, averaging 2.30×. This huge speedup, achieved on a single-issue processor with the **ptlu** instruction, is even 15% better than the speedup obtained on an 8-way superscalar processor (with 1 memory port, Table 6).

Without the **ptlu** instruction, wordsize scaling up to 64-bit and 128-bit words provides no performance improvement over the 32-bit words. This is mainly because table lookups cannot be efficiently parallelized when memory accesses are performed using standard **load** instructions, which can only read a single table entry at a time. As a result, table lookups are still implemented serially on the wider processors. While other sections of the code may sometimes be parallelized (for example effective address computations), these gains are offset by the overhead instructions used to convert between parallel and serial representations. Hence, the ciphers are essentially implemented in the same way on the 64-bit and 128-bit processors as on the 32-bit processor, i.e. without utilizing subword parallelism, resulting in the flat speedups shown in the columns in Table 7 labeled "w/o **ptlu**". In contrast, huge speedups are obtained when the **ptlu** instruction is used to parallelize the table lookups. With a 128-bit wordsize, speedups of 2.18× for MARS and 6.15× for AES-256 are obtained.

5. Past work

Inclusion of a dedicated instruction to accelerate table lookups in symmetric-key ciphers have previously been proposed in [3], [7], and [20]. For example, the **sbox** instruction in [3] can perform very fast lookups of tables located in main memory by accelerating the effective address computations. However, this solution has several restrictions; for example: (1) only 32-bit words can be read from a table, (2) each table must be aligned to 1024-byte boundaries, and (3) only one table is read by each **sbox** instruction. Furthermore, cache misses are still possible since tables are stored in main memory. In contrast, the **ptlu** instruction proposed in this paper can read variable-size data from a table entry, has no alignment restrictions, and can perform parallel reads from multiple tables. Also, cache misses are completely eliminated since dedicated on-chip tables are used.

The CryptoManiac processor [20] includes four 1 kB caches on the processor chip, which can be accessed with an **sbox** instruction. This differs from our **ptlu** proposal because only a single cache can be read with each **sbox** instruction rather than multiple caches in parallel. To read all four caches simultaneously, multiple-issue techniques such as 4-way VLIW is needed in CryptoManiac. In contrast, we allow up to eight tables to be read in parallel on a single-issue processor using a single **ptlu** instruction.

We defined an earlier version of the **ptlu** instruction in our previous work on the PAX cryptographic processor [7][16]. Compared to this, the new **ptlu** instruction presented in this paper is more versatile. For example, the previous **ptlu** instruction always required tables as wide as the processor wordsize, whereas our new proposal allows implementing narrower tables. Second, the **ptlu** instruction in PAX was used to select one of the eight on-chip tables and perform up to four simultaneous lookups from it. This requires using SRAM cells with four read ports, which increases the table area and also the access time. In contrast, the new **ptlu** instruction requires only single-ported standard SRAM cells. Using CACTI [5], we estimate that the new implementation requires less than 20% of the table area of PAX in [7], and is 14% faster in terms of access time.

6. Conclusions

The first contribution of this paper is the workload characterization of five widely-used symmetric-key ciphers that rely heavily on table lookups in their round structures. Our simulations indicate that all of these ciphers spend the largest fraction of their execution time during table lookups (up to 72% for AES).

Second, we describe a new **ptlu** instruction to accelerate and parallelize these table lookup. The area cost of this instruction is no greater than today's L1 caches (and likely to be smaller) and its latency is less than that of an ALU. Hence, it can be executed in a single cycle, with no cycle time impact to the base processor.

Our third contribution is to demonstrate that while *wordsize scaling* using standard **load** instructions is not very useful for symmetric-key ciphers, its effectiveness increases significantly if the **ptlu** instruction is added to the ISA. For example, the speedup obtained with **ptlu** for AES-256 increased from 2.32× on the 32-bit processor, to 6.15× on the 128-bit processor. Overall, we showed that the performance benefits of using **ptlu** together with *wordsize scaling* is far higher than increasing the number of instructions executed per cycle (IPC scaling) in superscalar or VLIW execution.

References

[1] Advanced Encryption Standard (AES), FIPS 197, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, Nov. 2001.

[2] Advanced Encryption Standard (AES) Development Effort, NIST, <http://csrc.nist.gov/CryptoToolkit/aes/index2.html>, Jan. 1997-Nov. 2001.

[3] J. Burke, J. McDonald, and T. Austin, "Architectural Support for Fast Symmetric-Key Cryptography", *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*), pp. 178-189, Nov. 2000.

[4] C. Burwick, et al, "MARS – A Candidate Cipher For AES", <http://www.research.ibm.com/security/mars.pdf>, Sept. 1999.

[5] CACTI, Compaq – Western Research Lab, <http://research.compaq.com/wrl/people/jouppi/CACTI.html>.

[6] A.M. Fiskiran and R.B. Lee, "Evaluating Instruction Set Extensions for Fast Arithmetic on Binary Finite Fields", *Proc. Int. Conf. Application-Specific Systems, Architectures, and Processors (ASAP)*, pp. 125-136, Sep. 2004.

[7] A.M. Fiskiran and R.B. Lee, "PAX: A Datapath-Scalable Minimalist Cryptographic Processor for Mobile Environments", *Embedded Cryptographic Hardware: Design and Security, Nadia Nedjah and Luiza de Macedo Mourelle, eds.,* Nova Science, NY, ISBN 1-59454-145-0, Sep. 2004.

[8] A.M. Fiskiran and R.B. Lee, "Performance Impact of Addressing Modes on Encryption Algorithms", *Proc. Int. Conf. Computer Design* (*ICCD*), *pp. 542-545*, Sep. 2001.

[9] A.M. Fiskiran and R.B. Lee, "Performance Scaling of Cryptography Algorithms in Servers and Mobile Clients", *Proc. Workshop on Building Block Engine Architectures for Computer Networks (BEACON),* Oct. 2004.

[10] IEEE 802.11 Wireless LAN Standards, IEEE 802.11 Working Group, <http://grouper.ieee.org/groups/802/11/>.

[11] R.B. Lee, A.M. Fiskiran, Z. Shi, and X. Yang, "Refining Instruction Set Architecture for High-Performance Multimedia Processing in Constrained Environments", *Proc. Int. Conf. Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 253-264, Jul. 2002.

[12] R.B. Lee and A.M. Fiskiran, "PLX: An Instruction Set Architecture and Testbed For Multimedia Information Processing", to appear in the *Journal of VLSI Signal Processing*.

[13] R.B. Lee and A.M. Fiskiran, "Multimedia Instructions in Microprocessors for Native Signal Processing", *Programmable Digital Signal Processors, Yu Hen Hu, ed.*, Marcel Dekker, pp. 91-145, Dec. 2001.

[14] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Oct. 1996.

[15] MIPS32 Architecture for Programmers Volume 2: The MIPS32 Instruction Set, v2.00, MIPS, available at <http://www.mips.com>.

[16] PAX Project, Princeton Architecture Laboratory for Multimedia and Security (PALMS), <http://palms.ee.princeton.edu/PAX>.

[17] PLX Project, Princeton Architecture Laboratory for Multimedia and Security (PALMS), <http://palms.ee.princeton.edu/PLX>.

[18] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley and Sons, 1996.

[19] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish: A 128-bit Block Cipher," <http://www.schneier.com/twofish.html>, Jun. 1998.

[20] L. Wu, C. Weaver, and T. Austin, "CryptoManiac: A Fast Flexible Architecture for Secure Communication", *Proc. Annual Int. Symposium on Computer Architecture* (*ISCA*), pp. 110-119, Jun. 2001.