On-Chip Lookup Tables for Fast Symmetric-Key Encryption

A. Murat Fiskiran and Ruby B. Lee
Princeton Architecture Laboratory for Multimedia and Security (PALMS)
Department of Electrical Engineering, Princeton University
{fiskiran, rblee}@princeton.edu

Abstract

On public communication networks such as the Internet, data confidentiality can be provided by symmetric-key ciphers. One of the most common operations used in symmetric-key ciphers are table lookups. These frequently constitute the largest fraction of the execution time when the ciphers are implemented using a typical RISC-like instruction set. To accelerate these table lookups, we describe a new hardware module, called PTLU (for Parallel Table Lookup*), which consists of multiple lookup tables that can be accessed in parallel. A novel combinational circuit included in the module can optionally perform simple logic operations on the data read from the tables. On a single-issue 64-bit RISC processor, PTLU provides maximum speedups of 7.7× for AES and 5.4× for DES. With wordsize scaling, PTLU speedups are significantly higher than that available through more conventional architectural techniques such as superscalar or VLIW execution.*

1. Introduction

Symmetric-key cryptography can be used to provide data confidentiality [18] on public communication networks such as the Internet. This involves encrypting a *plaintext* message *P* using a symmetric-key algorithm (cipher) and a secret key *K*. The encrypted message (*ciphertext*) is then sent to the receiver, where it is decrypted using the same cipher and secret key.

Symmetric-key ciphers usually have an *iterated round* structure, where a short sequence of operations (called a *round*) is repeated on the plaintext block to compute the ciphertext [18]. The input of a round consists of the output of the previous round and one or more subkeys, which are derived from the secret key. Common round operations include table lookups, modular addition (subtraction), logical operations, shifts, rotates, multiplication, and bit permutations [18][22]. On a programmable processor that implements a RISC-like instruction set, table lookups generally consume the greatest fraction of the execution time [8]. In this paper, we describe how these can be accelerated cost-effectively with a new *Parallel Table Lookup* (PTLU) Module.

The rest of the paper is organized as follows. In Section 2, we study the workload characteristics of six representative symmetric-key ciphers. Section 3 describes the PTLU hardware and Section 4 discusses its area cost and cycle time impact. In Section 5, we discuss the performance of PTLU. Section 6 reviews the related past work and Section 7 is the conclusion.

2. Cipher suite

Table 1 lists the symmetric-key ciphers we selected for this study. For each cipher, we show the *block size*, typical *key size*, and the number of rounds. Block size is the amount of data that the cipher can encrypt at a time, and key size relates to the strength of the cipher against cryptanalytic attacks [18].

Data Encryption Standard (DES) [18] and its variant 3DES were the NIST standards for block encryption from 1976 to 2001. 3DES continues to be used extensively in many systems. RC4 is a popular stream cipher [18], which is originally used in the IEEE 802.11 wireless standard. Blowfish [18] is used in many protocols and applications, for example GPG, SSH, SSLeay, and JAVA cryptography extensions [19]. Advanced Encryption Standard (AES) [1] is the current NIST standard for block encryption. Its key size can be 128, 192, or 256 bits. We denote these AES-128, AES-192, and AES-256 respectively. Twofish [21] and MARS [5] are two of the five finalist ciphers in the AES selection program [2]. Together with AES, these new ciphers can be said to represent the current thinking in symmetric-key cipher design.

This work is supported in part by NSF CCR-0326372 and by DoD.

Proc. IEEE 16th Int. Conf. Application-Specific Systems, Architectures, and Processors (ASAP), pp. 356-363, Jul. 2005.

Table 1: Cipher suite

Cipher	Block Size (bits)	Key Size (bits)	Num. Rounds	Num. Tables	Table Structure	Num. Lookups
DES	64	56	16	8	$2^6 \times 32$	128
3DES	64	112	48	8	$2^6 \times 32$	384
RC4	8	128	1*	1	$2^8 \times 8$	3 + 2W
Blowfish	64	128	16	4	$2^8 \times 32$	64
AES-128	128	128	10	4	$2^8 \times 32$	160
AES-192	128	192	12	4	$2^8 \times 32$	192
AES-256	128	256	14	4	$2^8 \times 32$	224
Twofish	128	128	16	4	$2^8 \times 32$	128
MARS	128	128	32	2	$2^8 \times 32$	80

* RC4 does not have an iterated round structure; hence we show the number of rounds as 1. To generate each byte of keystream, RC4 requires 3 table reads and 2 writes.

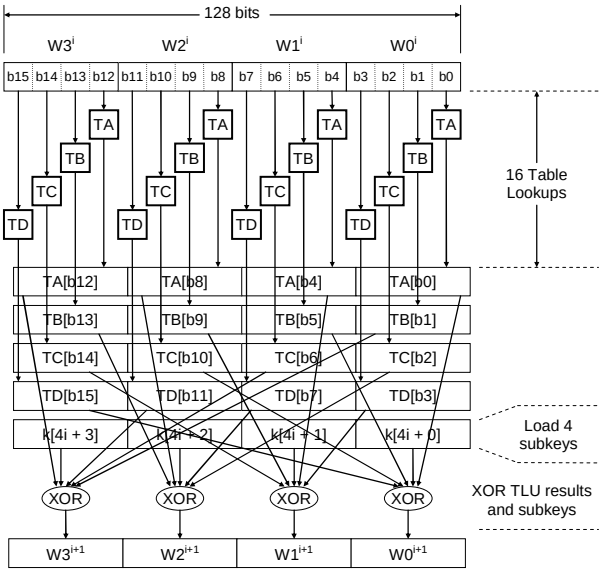


Figure 1: AES round

Table 2: Instruction set

	Class	Instructions
Base ISA	ALU	add, addi, sub, subi, and, andi, or, ori, xor, xori, not, loadi
	Shift	sra, srai, srl, srli, sll, slli, shrp (shift right pair)
	Multiply	mul.lo, mul.hi
	Memory	load, store (base+displacement addressing)
	Branch	beq, bne, bg, bge, call, return, trap
New ISA		**ptrd.x, ptrd.s, ptw1, ptwn, byte_perm**

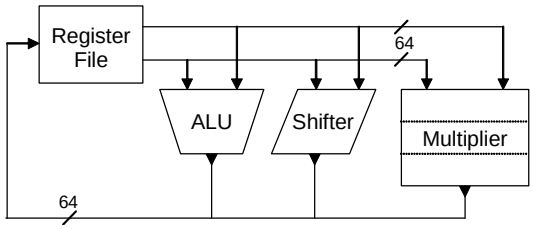


Figure 2: Baseline 64-bit single-issue processor

Table 3: Breakdown of cipher execution time

		DES	3DES	RC4	Blowfish	AES-128	AES-192	AES-256	Twofish	MARS
	Block size (bits)	64	64	8	64	128	128	128	128	128
	Cycles per block	1147	3384	18	408	870	1056	1272	1753	1677
% Execution Cycles Spent in ..	Table Lookups	38	44	54	36	72	72	72	43	34
	Arithmetic	-	-	14	26	-	-	-	15	10
	Logical	21	24	26	34	24	24	24	32	18
	Multiplication	-	-	-	-	-	-	-	-	19
	Fixed shift/rotate	8	9	-	-	-	-	-	4	5
	Variable rotate	-	-	-	-	-	-	-	-	8
	Bit permutation	26	15	-	-	-	-	-	-	-
	Other	7	8	6	4	4	4	4	6	6

2.1. Table lookups in symmetric-key ciphers

We show the AES round structure in Figure 1 to illustrate how table lookups are typically used in symmetric-key ciphers. In Table 1, we also summarize the number and structure of the lookup tables used by each cipher. The notation $2^a \times b$ is used to denote a table with 2^a entries, where each entry is b-bits wide.

In AES, the input to the i^{th} round is a 128-bit block composed of four 32-bit words. The bytes in these words are labeled b0 to b15. There are four $2^8 \times 32$ tables, labeled TA-TD. The rightmost byte of each word is used as index into TA; the next byte is used as index into TB; and so on, until all tables are accessed four times. The table lookup results and four subkeys are then exclusive-or'ed (XORed) as shown.

Of the remaining ciphers in our suite, Blowfish, MARS, and Twofish are similar to AES in that they use multiple $2^8 \times 32$ tables. DES and 3DES use eight $2^6 \times 32$ tables [18], while RC4 uses a single $2^8 \times 8$ table. In the next section, we measure the fraction of the execution cycles that each cipher spends in table lookups.

2.2. Execution time analysis

We use the PLX toolset [13][17] to perform workload analysis of each cipher. For baseline performance data, the ciphers are implemented using the RISC-like instruction set shown as the *Base ISA* in Table 2. Our assembly code follows the optimizations described in [10] and [18]. We configure the simulator to model the 64-bit single-issue processor shown in Figure 2. All instructions execute in a single cycle except multiplication, which has a 3-cycle latency. We also assume a perfect memory with single-cycle load/store instructions. Table 3 shows the simulation results, which includes: (a) the execution cycles used per block of encryption, (b) the round operations in each cipher, and (c) the fraction of the execution time consumed by these. Our data presented so far enable us to make the following observations:

- Table lookups consume the greatest fraction of the execution time for all ciphers, varying from 34% for MARS to 72% for AES (Table 3).
- Tables are few (at most eight) and have constant size. Except in RC4, all table accesses are reads (Table 1).
- Number of entries per table is small (at most 256) and the data read is either 8 or 32 bits (Table 1).

Furthermore, it is generally possible to perform the table lookups in parallel. For example, all 16 lookups in an AES round (Figure 1) can be fully parallelized, constrained only by hardware resources. Next, we describe how hardware support can be used to exploit these properties to accelerate symmetric-key ciphers.

3. Parallel Table Lookup Module (PTLU)

We propose adding an on-chip scratchpad memory to the baseline processor in Figure 2. Among other possible uses, it can be used for fast parallel table lookups, so we call this memory the *Parallel Table Lookup* (PTLU) module. On a 64-bit processor, this memory has up to 8 read ports, and can be implemented with up to 8 blocks of standard SRAM memory, each with a single read port. The extended datapath is shown in Figure 3 (the shifter and the multiplier are not shown for brevity).

The inputs to the PTLU module are two source registers; the output is one result register. To write to the register file, PTLU can use either the functional unit result bus or the cache memory bus. Figure 3 shows the former option.

Figure 4 shows the details of the PTLU module. There are eight tables with 256 entries each, where each entry is 32 bits wide. In co-designed embedded systems, the number and/or the width of the tables can be scaled down to limit cost and power. During a read, each table is accessed by an 8-bit index read from the first source register Rs1. The rightmost byte of Rs1 (B0) accesses T0; the next byte (B1) accesses T1; and so on. All eight tables can be read in parallel.

The eight 32-bit lookup results, one from each table, are then routed through a simple network of combinational logic, comprised of six XOR-Multiplexers (XMUX0 to XMUX6) and an XOR unit. The XOR unit simply XORs the output of XMUX6 with Rs2. The signals that control the XMUXs come from a decoder, which, in turn, is controlled by a sub-op from the instruction word.

The internal structure of the XMUXs is shown in Figure 5a. Each XMUX has two 32-bit inputs labeled L and R, for left and right. Based on the values of two control bits (C1, C0), the XMUX output can be: 0, L, R, or L XOR R. This is summarized in Table 4. XMUX6, which is shown in Figure 5b, is different in that its output is either L XOR R, or, $L \parallel R$, where $\parallel$ denotes concatenation.

3.1. Instructions for reading the PTLU module

We describe three *ptrd* (parallel table read) instructions to read the PTLU module. These can be added to a base instruction set such as the one shown in Table 2. The first instruction has the following format:

ptrd.x1 Rd, Rs1, Rs2

Here, Rd is the destination register; Rs1 is the first source register, which supplies the byte-sized table indices; and Rs2 is the second source register, which is routed to the right input of the XOR unit in Figure 4. The '*x1*' in the mnemonic indicates that this instruction *XORs* all 8 lookup results and Rs2 into a *single* value. This is achieved by setting all XMUX control bits to 1.

The second *ptrd* instruction has the following format:

ptrd.x2 Rd, Rs1, Rs2

Here, the table lookup results and Rs2 are *XORed* into to *two* parallel values. This is achieved by setting all XMUX control bits to 1 except for XMUX6, which has

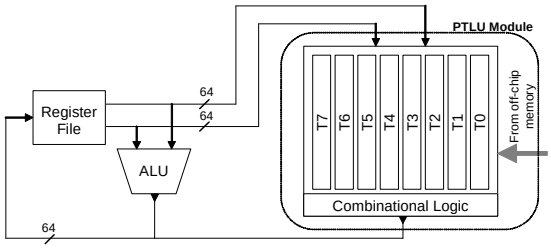


Figure 3: Processor with PTLU

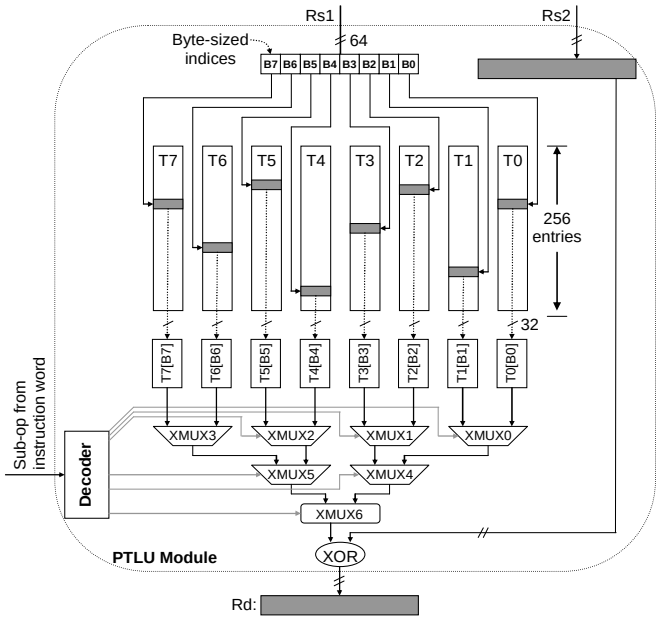
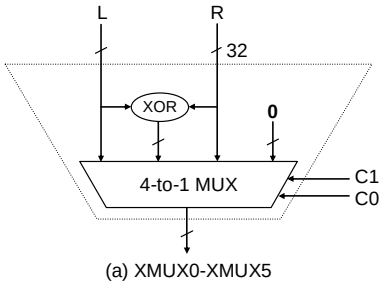
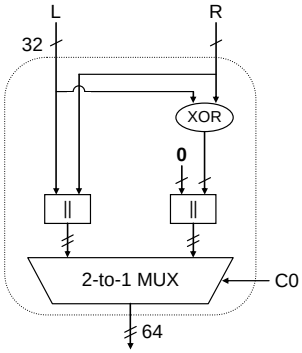


Figure 4: Reading of the PTLU module



(a) XMUX0-XMUX5



(b) XMUX6

Figure 5: Operation of the XMUXs

Table 4: XMUX output

	(C1, C0) Value			
	(1, 1)	(1, 0)	(0, 1)	(0, 0)
XMUXs 0 to 5	L ⊕ R	0	L	R
XMUX6	L ⊕ R	L ‖ R	N/A	N/A

its C0 bit set to zero. Hence, the output of XMUX6 becomes the concatenation of the outputs of XMUX5 and XMUX4.

To select and write a table lookup result to Rd without an XOR, we define the *ptrd.s* instruction (*s* signifies *select*):

ptrd.s.tab Rd, Rs1, Rs2

Here, *tab* is a 2-bit sub-op field that selects one of T0-T3 to write to the rightmost 32 bits of Rd. Simultaneously, a second table is selected from T4-T7 to write to the leftmost 32 bits of Rd. For example, if *tab* = 2, then the output of T2 and T6 are written to the rightmost and leftmost 32 bits of Rd respectively, hence implementing two parallel table lookups.

3.2. Instructions for writing the PTLU module

To write the tables in the PTLU module, we define two *ptw* (parallel table write) instructions:

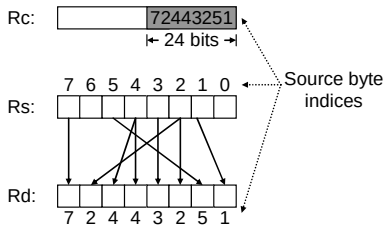
ptw1.table Rs1, Rs2

The '*1*' in the instruction signifies that a *single* PTLU table is written, which is selected by the 3-bit sub-op field *table*. The rightmost byte of Rs1 is used as the table index and the selected table entry is written with the rightmost 32 bits of Rs2.

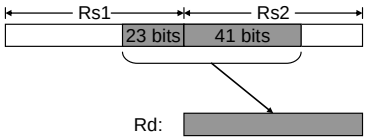
Ptw1 can only write one table at a time, but this does not degrade encryption performance since ciphers do not require parallel table writes. However, fast parallel writes may be desired for rapid initialization of tables at setup time. For this, we define the *ptwn* instruction (*n* signifies that multiple tables are written in parallel). *Ptwn* uses the wide memory bus shown on the right in Figure 3 and has the following format:

ptwn Rs, Rb, disp

Here Rb is a base address register and *disp* is the displacement. The 32-byte memory block from address Rb + *disp* is written to a common row of all eight PTLU tables in parallel. The row is selected by the rightmost byte of Rs. All PTLU entries can be written using 256 *ptwn* instructions.



(a) byte_perm Rd, Rs, Rc



(b) shrp Rd, Rs1, Rs2, 23

Figure 6: *Byte_perm* and *shrp* instructions

3.3. Instructions for rearranging index bytes

In the *ptrd* instruction, the source bytes in Rs1 access the PTLU tables in a fixed order. To allow lookups where the source bytes need to access T0-T7 in random order, we propose adding a *byte_perm* (byte permutation) instruction to the Base ISA:

byte_perm Rd, Rs, Rc

This can perform any permutation of the bytes in Rs based on the control bits in Rc. By complementing PTLU with *byte_perm*, a much wider variety of parallel lookups can be realized. An example of *byte_perm* is given in Figure 6a. The rightmost 3 bits in Rc choose the source byte to be written to the rightmost byte of Rd (B0); the next 3 bits choose the byte to be written to B1; and so on. The leftmost 40 bits of Rc are unused. This is like the *permute* instruction in MAX-2 [15] and the *pperm* instruction in [16], but restricted to bytes.

Byte_perm can be implemented in hardware using eight 8-to-1 multiplexers (each 8-bit-wide). In this paper, we implement *byte_perm* by extending the shifter, which we call the *Shift-Permute Unit* (SPU). To permute more than eight bytes efficiently, *byte_perm* can be used together with the *shift right pair* (*shrp*) instruction, which is shown in Figure 6b. We will give an example of this in Section 5.

4. Area cost and delay analysis

To evaluate the cost of new hardware, we first establish baseline results by designing in VHDL the functional units of the processor in Figure 2, which implements the Base ISA in Table 2. We then extend this processor with the new instructions we proposed. For area and delay estimates, we perform gate-level synthesis of the functional units using Synopsys tools with TSMC 90nm technology library. For the PTLU module, we use CACTI 3.2, which is a tool for estimating the access time, area, and aspect ratio of memory components [6].

Table 5 summarizes our results. For each functional unit, we report absolute area in square-microns, the equivalent number of minimum-sized two-input NAND gates, and relative area normalized to the ALU. Delay is given as absolute delay in nanoseconds, relative delay with respect to ALU, and number of clock cycles assuming that ALU latency is a single cycle.

We verify that implementing *byte_perm* in the modified shifter does not impact cycle time or increase the shifter latency in terms of clock cycles. The access time of the PTLU tables is 67% of the ALU delay. The XMUX tree could be synthesized so that the total delay through the PTLU module is no greater than the ALU delay. Hence *ptrd* and *ptw* instructions have single cycle latency. Of the total area of the PTLU module, 90.5% is consumed by the eight lookup tables and 9.5% is consumed by the XMUXs.

In today's high-end embedded processors, for example Intel PXA270 [11], the size of the on-chip data cache is typically about 32 kB. The PXA270 also includes an additional 256 kB SRAM to be used as scratchpad memory. Compared to these, the size of the PTLU module is small (Table 5); about 35% of the 32 kB cache and 5% of the 256 kB cache.

5. Performance

5.1. Optimized AES

To illustrate the use of PTLU and *byte_perm* instructions, Figure 7 shows the optimized assembly code for AES on a 64-bit processor. Figure 8 shows the data flow in the first half of the code. The 128-bit AES state (refer to Figure 1) is supplied in two 64-bit registers (R11, R10). The PTLU tables are initialized to two sets of the four AES tables. The first four *byte_perm* and *shrp* instructions permute (R11, R10) such that R14 contains eight indices into tables whose results can be directly XORed. These are the bytes (3,14,9,4) and (15,10,5,0) in Figure 1. The *load.8* instruction loads the first two round subkeys into R15. The *ptrd.x2* instruction performs eight lookups using the bytes in R14. These results are XORed in pairs by XMUX0–XMUX5. Next, XMUX6 concatenates the output of XMUX5 and XMUX4. The result is then XORed with the subkeys in R15. Destination register R10 then contains ($W1^{i+1}$,$W0^{i+1}$). The last four instructions similarly compute ($W3^{i+1}$,$W2^{i+1}$). The entire AES round takes only 10 instructions.

Table 5: Area and delay of baseline and enhanced functional units

Functional Unit / Component	Area			Delay		
	μ^2	**NAND Gate Equivalent**	**Normalized** (ALU = 1.00)	**ns**	**Normalized** (ALU = 1.00)	**Cycles**
ALU	19122	7904	1.00	0.55	1.00	1
Shifter	6660	2753	0.35	0.45	0.82	1
SPU with *byte_perm*	7432	3512	0.44	0.55	1.00	1
PTLU: 8 Tables	322464	133296	16.86	0.37	0.67	1
PTLU: XMUX Tree	33972	14043	1.78	0.18	0.33	1
PTLU: Total	356436	147336	18.64	0.55	1.00	1
32 kB 2-way cache w/ 64-byte blocks	1012722	418619	52.96	0.63	1.15	2
256 kB 2-way cache w/ 64-byte blocks	6913820	2857895	361.58	0.88	1.60	2

```
# R11 contains bytes 15-8 of AES state, R10 contains bytes 7-0 of AES state

byte_perm R12, R10, R1        # Config. reg. R1 contains byte indices (7,6,2,1,5,4,3,0)
                              # R12 receives bytes (7,6,2,1,5,4,3,0) of AES state
byte_perm R13, R11, R1        # R13 receives bytes (15,14,10,9,13,12,11,8) of AES state
shrp      R14, R12, R13, 32   # R14 receives bytes (5,4,3,0,15,14,10,9) of AES state
byte_perm R14, R14, R2        # Config. reg. R2 contains byte indices (5,2,0,6,3,1,7,4)
                              # R14 receives bytes (3,14,9,4,15,10,5,0) of AES state

load.8    R15, R9, 0          # Load 2 subkeys; R9 is base address of subkey array

ptrd.x2   R10, R14, R15       # Lookup 8 tables, XOR results and round subkey;
                              # PTLU tables (T3-T0) and (T7-T4) contain AES tables (TD-TA)

# Now repeat for the remaining 8 lookups
shrp      R14, R13, R12, 32   # R14 receives bytes (13,12,11,8,7,6,2,1) of AES state
byte_perm R14, R14, R2        # R14 receives bytes (11,6,1,12,7,2,13,8) of AES state
load.8    R15, R9, 8          # Load next two subkeys
ptrd.x2   R11, R14, R15       # R11, R10 contain the new AES state
```

Figure 7: Optimized AES round with ptrd (64-bit processor)

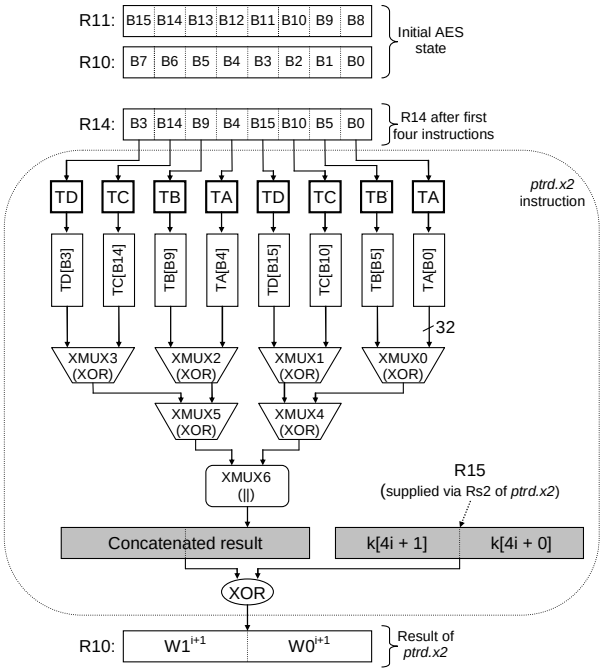


Figure 8: Data flow in the optimized AES round

Table 6: PTLU performance
(64-bit single-issue processor)

Cipher	Block size (bits)	Cycles with Base ISA	Speedup with PTLU
DES	64	1147	5.41 ×
3DES	64	3384	5.32
RC4	8	18	2.00
Blowfish	64	408	1.66
AES-128	128	870	6.91
AES-192	128	1056	7.23
AES-256	128	1272	7.66
Twofish	128	1753	2.81
MARS	128	1677	1.23

Table 7: Superscalar performance vs. PTLU

	Speedup w/ Superscalar Execution (Base ISA)							Speedup w/ PTLU		
Cipher	**1/1**	**2/1**	**2/2**	**4/1**	**4/2**	**8/1**	**8/2**	**32-bit**	**64-bit**	**128-bit**
3DES	1.00	1.62	1.85	1.78	2.32	1.88	2.73	3.41	5.32	5.32
AES-128	1.00	1.58	1.71	1.85	2.23	2.02	2.49	2.79	6.91	27.19

5.2. Results and discussion

Baseline vs. PTLU: Table 6 summarizes the performance improvement for all ciphers on the 64-bit single-issue processor with PTLU. The speedups are relative to the execution cycles per block of encryption with the Base ISA in Table 2. While all ciphers benefit from the new instructions, some show huge performance gains. The speedups for DES, 3DES, and AES range from 5.3× to 7.7×. The remaining ciphers have speedups varying from 1.2× for MARS to 2.8× for Twofish.

Multiple-issue (without PTLU): For 3DES and AES-128, Table 7 shows the speedups obtained with superscalar execution on processors with issue widths from 1 to 8. Speedups are relative to a single-issue 32-bit processor that implements the 32-bit version of the Base ISA in Table 2. In the notation *a/b*, *a* is the issue width and *b* is the number of memory ports. Superscalar execution provides significant speedups for both ciphers; up to 1.9× for 2-way and 2.3× for 4-way. Further increasing the issue width to 8 provides only minor additional performance (up to 2.7×).

Wordsize scaling vs. superscalar: The last 3 columns of Table 7 show the speedups when PTLU is added to single-issue 32-bit, 64-bit, and 128-bit processors. On the 32-bit processor, PTLU is implemented with four 2^8×32 tables, so it can be compared to a scratchpad memory with four read ports. Similarly, PTLU on the 128-bit processor uses 16 2^8×32 tables, and works like a memory with 16 read ports. The XMUX tree is scaled accordingly.

While comparing single-issue processors with and without PTLU, we assume that the 64-bit and 128-bit processors support *subword parallelism* [14][15], which involves partitioning the datapath into units smaller than a word, called *subwords*. Multiple subwords packed in a word can be processed in parallel using *subword parallel instructions*. For example, four pairs of 32-bit subwords packed in two source registers can be added with a single *parallel add* (*padd*) instruction on the 128-bit processor. We assume that parallel versions of all ALU and shift instructions in Table 2 are supported for 32-bit subwords.

On a single-issue 32-bit processor, PTLU provides 3.4× and 2.8× speedup for 3DES and AES respectively. Both figures are better than the speedups obtained on an 8-way superscalar processor without PTLU. On the 64-bit processor, PTLU speedup increases to 6.9× for AES. This should be compared to the 1.7× speedup of the 2-way 32-bit processor since both have equivalent degrees of operand parallelism. Similarly, the 27.2× speedup on the single-issue 128-bit processor can be compared to the 2.2× speedup of the 4-way 32-bit processor. These results clearly indicate that using PTLU with wider processors is far more effective for improving performance than increasing the issue width in superscalar processors. Compared to a multi-issue processor, a wider single-issue processor offers savings in register ports, data buses, bypass paths, and instruction issue logic [14].

Other processors: In Table 8, we compare the AES-128 performance of several programmable processors. Compared to the popular ARM9 embedded processors [3], a 32-bit baseline processor with PTLU (PTLU-32) provides 5.6× better performance. A single-issue PTLU-64 easily outperforms more complicated multi-way processors like Pentium III, IA-64, and PA-8200 [3][20]. A single-issue PTLU-128 provides 2.8× better performance than CryptoManiac [23], which is 4-way VLIW (*Very Long Instruction Word*). The 32-cycle latency of PTLU-128 is only 22 cycles more than a hardwired AES chip [12].

Table 8: AES-128 performance

Platform	Reference	Cycles
ARM9TDMI	Bertoni [3][10]	1764
Pentium III	Gladman [3][10]	381
IA-64	Schneier [20]	190
HP PA-8200	Schneier [20]	280
CryptoManiac	Wu [23]	90
AES ASIC	Kuo [12]	10
32-bit PTLU	This paper	315
64-bit PTLU		126
128-bit PTLU		32

6. Past work

Dedicated instructions to accelerate table lookups in symmetric-key ciphers have previously been used in [4] and [23]. The *sbox* instruction in [4] performs fast lookups of tables located in main memory by accelerating the effective address computations. The CryptoManiac processor [23] uses a similar *sbox* instruction to read its four 1 kB on-chip caches. Both approaches differ from our PTLU proposal because only a single table can be read with each *sbox* instruction rather than multiple tables in parallel. In contrast, we allow up to eight tables read in parallel on a 64-bit single-issue processor using a single *ptrd* instruction.

In our earlier work on the PAX crypto-processor [9] and more recently in [7], we described how on-chip lookup tables can be used to accelerate symmetric-key encryption. Our work in this paper is different in three important ways, resulting in much higher speedups. First, both the number of tables and table width are

fully scalable in the PTLU module. Furthermore, each PTLU table has a single read port, hence can be implemented with standard SRAM cells. This differs from our proposal in [9], where each lookup table has four read ports (hence larger and slower) and is always as wide as the processor wordsize.

Second, the *parallel lookup* instructions in [7] and [9] use multiple sub-op fields to specify the number of lookups to be performed, data size, and the index bytes to be used. In contrast, the new *ptrd* instructions provide this information implicitly; hence they are simpler, without any loss of performance or flexibility.

Compared to [7] and [9], the most distinctive feature of the new PTLU module in this paper is the novel XMUX tree, which optionally performs simple logic operations on table data. The XMUX tree increases the table area by only 11% compared to [7], but provides much higher performance, for example an *additional* 4.6× speedup for AES-128 on the 128-bit processor.

7. Conclusions

The first contribution of this paper is the workload characterization of six representative symmetric-key ciphers. We show that all these ciphers spend the largest fraction of their execution time in table lookups (up to 72% for AES).

Second, we describe a new PTLU module to accelerate these table lookups. This module is smaller than today's L1 caches and its latency is a single cycle. We also describe how a *byte_perm* instruction can complement the PTLU module to allow a greater variety of parallel table lookups. On the 64-bit single-issue processor, the new instructions generate huge speedups for most of the ciphers, up to 7.7× for AES-256. An extra benefit of PTLU is the elimination of cache read misses in table lookups. Unlike a *load* instruction which can take either a single cycle (cache hit) or many cycles (cache miss), a *ptrd* instruction always takes one cycle. This eliminates the variability in encryption latency, thwarting some cipher attacks.

Our third contribution is to show that the effectiveness of PTLU increases significantly as the wordsize of the processor increases. PTLU speedups obtained on wider processors are far higher than those obtained by increasing the number of instructions executed per cycle in superscalar or VLIW processors.

10. References

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

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

[3] G. Bertoni et al., "Efficient Software Implementation of AES on 32-bit Platforms", *Lecture Notes In Computer Science*, vol. 2523, Springer-Verlag, pp. 159-171, 2003.

[4] 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.

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

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

[7] A.M. Fiskiran and R.B. Lee, "Fast Parallel Table Lookups to Accelerate Symmetric-Key Cryptography", *Proc. ITCC, Embedded Cryptographic Systems*, pp. 526-531, Apr. 2005.

[8] 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.

[9] A.M. Fiskiran and R.B. Lee, "PAX: A Datapath-Scalable Minimalist Cryptographic Processor for Mobile Environments", *Embedded Cryptographic Hardware: Design and Security*, Nova Science, NY, Sep. 2004.

[10] B. Gladman, AES Second Round Implementation Experience, source code for AES finalists available at <http://fp.gladman.plus.com/cryptography_technology/aesr2>.

[11] Intel PXA270 Processor for Embedded Computing – Product Brief, Intel, Doc. ID 302302-001, at <http://www.intel.com>.

[12] H. Kuo and I. Verbauwhede, "Architectural Optimization for a 1.82 Gb/s VLSI Implementation of the AES Rijndael Algoritm", *Lecture Notes in Computer Science*, vol. 2162, pp. 51-64, May. 2001.

[13] R.B. Lee and A.M. Fiskiran, "PLX: An Instruction Set Architecture and Testbed For Multimedia Information Processing", *Journal of VLSI Signal Processing*. vol. 40, no. 1, pp. 85-108, May 2005.

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

[15] R.B. Lee, "Subword Parallelism with MAX-2", *IEEE Micro*, vol. 16, no. 4, pp. 51-59, Aug. 1996.

[16] R.B. Lee, Z. Shi, and X. Yang, "Efficient Permutation Instructions for Fast Software Cryptography", *IEEE Micro*, vol. 21, no. 6, pp. 56-69, Dec. 2001.

[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, The Blowfish Encryption Algorithm, <http://www.schneier.com/blowfish.html>.

[20] B. Schneier and D. Whiting, "A Performance Comparison of the Five AES Finalists", *Proc. Third AES Conference*, pp. 123-135, Apr. 2000.

[21] 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.

[22] Z. Shi, X. Yang, and R.B. Lee, "Arbitrary Bit Permutations in One or Two Cycles", *Proc. IEEE Int. Conf. Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 237-247, Jun. 2003.

[23] 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.