Performance Scaling of Cryptography Operations in Servers and Mobile Clients

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

Abstract

Cryptography algorithms are essential building blocks used to provide security on public communication networks such as the Internet. Concurrent with the increases in wireless connectivity and data rates, security protocols have been expanded over the past years to include more resource-friendly cryptography algorithms such as Elliptic Curve Cryptography (ECC). In this paper, we first describe and characterize these newer security algorithms suitable for mobile environments. We consider symmetric-key, hash, public-key, ECC, and digital signature algorithms. Next, we describe new architectural techniques to accelerate these algorithms. We focus on the table lookup operations used in the symmetric-key ciphers and the multi-precision arithmetic operations used in the public-key ciphers. Our third contribution is to show how: (1) the performance of a server can be scaled to support a growing number of mobile clients, and (2) the performance of a client can be scaled to meet particular wireless data transfer rates.

1. Introduction

Cryptography algorithms are the building blocks used to enable secure communications on public data channels such as the Internet. Due to the compute-intensive nature of these algorithms, high-performance cryptography is particularly difficult for mobile wireless devices, where the computational and memory resources are very constrained and the network data rates may be as high as 100 Mbps (e.g. UWB wireless technology [1]).

To facilitate cryptography processing in constrained environments, resource-efficient algorithms have been added to security protocols. Two examples are *Elliptic Curve Cryptography* (ECC) [2]-[4] and the *Advanced Encryption Standard* (AES) [5]. Our first contribution in this paper is the selection and workload characterization of a cryptography suite that includes these newer algorithms. This suite, which we call *mCrypt,* is representative of the next-generation security processing workload as it includes the cryptography algorithms frequently used by both servers and mobile clients.

Our second contribution is the description of instruction set architecture (ISA) methods to accelerate the symmetric-key and public-key algorithms in the *mCrypt* suite. These methods are *scalable* so that they are applicable in platforms with varying degrees of computational resources such as sensors, smartcards, cell phones, notebooks, desktops, and servers.

As our third contribution, we show how: (1) the performance of a server can be scaled to support a growing number of clients, and (2) the performance of a client can be scaled to meet particular data transfer rates.

The rest of the paper is organized as follows. In Section 2, we describe the algorithms in the mCrypt suite. In Section 3, we study the symmetric-key algorithms, and describe how to accelerate the table lookup operations frequently used by these. In Section 4, we study the public-key algorithms and the impact of wordsize scaling on performance. In Section 5, we show how client and server performance can be scaled to meet varying performance targets. Section 6 concludes the paper.

2. mCrypt suite of cryptography algorithms

A complete cipher suite includes *symmetric-key*, *hash*, and *public-key* algorithms [6]. Each of these classes fulfills a different security function.

Symmetric-key algorithms ensure the confidentiality of messages exchanged on a public network. *Hash algorithms* provide data integrity by verifying that the messages exchanged on the network arrive at their destinations unaltered. *Public-key algorithms* are used for authentication and digital signatures.

A subgroup of public-key algorithms is the Elliptic-Curve Cryptography (ECC), which offers higher security per key bit compared to the previous generations of public-key algorithms, so that shorter keys can be used to attain the same level of security. (For example, 160-bit ECC has the same security as 1024-bit RSA [7].) Smaller keys permit faster computations and also require less

This work was supported in part by Kodak (A.M. Fiskiran is a Kodak Fellow) and by NSF research grants CCR-0326372 and CCR-0105677.

A. Murat Fiskiran and Ruby B. Lee, "Performance Scaling of Cryptography Operations in Servers and Mobile Clients", *Proceedings of the Workshop on Building Block Engine Architectures for Computer Networks (BEACON),* October 2004.

Table 1: Symmetric-key ciphers in mCrypt

Name	Block Size / Key Size (bits)[1]	Description
AES [5]	128 / 128	US Federal standard for block encryption since 2001.
DES 3DES [6]	64 / 56 64 / 112	Former standards for block encryption.
IDEA, Blowfish [6]	64 / 128	Used in OpenSSL, WTLS, GPG and SSH.
RC4[2] [6]	8 / 128	Used in WEP - part of the WLAN standard IEEE 802.11 [8].
RC5 [6]	64 / 128	Used in the WTLS security protocol specified in WAP [9].
RC6, MARS, Twofish, Serpent	128 / 128	AES finalists [10].

Table 2: Hash algorithms in mCrypt

Name	Block size / Hash size (bits)	Description
SHA-1	512 / 160	Specified in SHS [11]. SHA-1 is also a used as a subroutine in DSS [4].
SHA-256	512 / 256	
SHA-384	1024 / 384	
SHA-512	1024 / 512	
MD5 [6]	512 / 128	General use for integrity verification (e.g. **md5sum** utility).

[1] For the ciphers that use variable length blocks (e.g. RC4, RC5) and/or variable length keys (e.g. AES, RC4, RC5, Blowfish) a typical value for the block/key size is shown.
[2] Due to security vulnerabilities in WEP, a higher-layer security protocol is used in addition to RC4.

Table 3[1]: Basic operations in symmetric-key block ciphers and hash algorithms

	AES	DES, 3DES	RC4	RC5	RC6	IDEA	Blowfish	MARS	Twofish	Serpent[2]	SHA-1	SHA-256	SHA-384	SHA-512	MD5
Add / Subtract			•	•	•	•	•	•	•		•	•	•	•	•
Logical	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Table Lookup (TLU)	•	•	•				•	•	•						
Integer Mult.					•	•		•							
Binary Mult.									•						
Shift										•		•	•	•	
Fixed rotate		•		•	•			•	•	•	•	•	•	•	•
Variable rotate				•	•			•							
Permutation		•							•						

[1] Only the operations that have cryptographic significance are listed.
[2] The reference implementation of Serpent includes table lookups but in practice these are realized as a sequence of logical operations.

Table 4: Structure of lookups tables

Algorithm	Block size (bits)	Table Structure[1]	Lookups Per Block of Encryption
AES	128	(4, 8, 32)	160
DES	64	(8, 6, 32)	128
3DES	64	(8, 6, 32)	384
RC4	8	(1, 8, 8)	Reads = 3, Writes = 2.
Blowfish	64	(4, 8, 32)	64
MARS	128	(2, 8, 32)	56
Twofish	128	(4, 8, 8)	48

[1] Shows (*# tables*, *# index bits, # bits in a table entry*).

Table 5: Breakdown of execution time

	Algorithm / %					
Operations	AES	DES, 3DES	RC4	Blowfish	MARS	Twofish
Add / Subtract			14	28	10	5
Logical	28	18	28	35	18	7
Table Lookup (TLU)	72	35	58	37	34	24
Integer Mult.					25	
Binary Mult.						54
Shift						
Fixed rotate		15			5	10
Variable rotate					8	
Permutation		32				0
Total %	100	100	100	100	100	100

storage, which is important for very resource-limited platforms such as sensors. Due to its suitability for constrained environments, ECC was added to major security protocols such as the TLS/SSL [12], WTLS [9], and DSS [4].

Tables 1 and 2 list the symmetric-key and hash algorithms in mCrypt and provide references to the algorithm definitions. In general, each algorithm is either an important security standard by itself (e.g. AES) or is part of a larger standard (e.g. SHA-1 in SHS [11]).

3. Accelerating symmetric-key algorithms

3.1. Workload characteristics

For workload characterization of the mCrypt algorithms, we simulate their optimized implementations using the PLX [13] and the SimpleScalar toolsets [14]. Table 3 summarizes the main operations used by each symmetric-key algorithm. Table 4 summarizes the structure of the lookup tables used by these ciphers. In Table 5, we show a breakdown of execution times of the ciphers that use table lookups. From these results, we observe the following:

- Six of the 10 ciphers use table lookups; and 5 of these spend the largest fraction of their execution time during these table lookups.

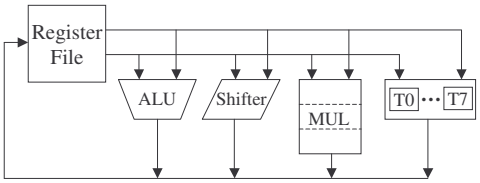
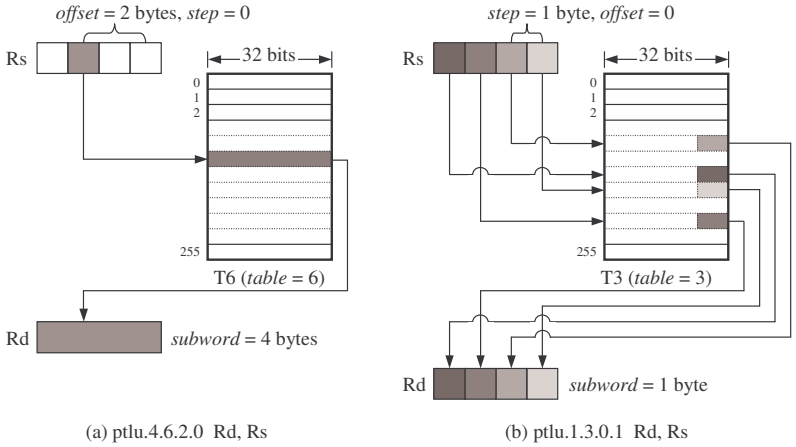


Figure 1: Single-issue PAX processor

Table 6: Speedups with ptlu

Algorithm	Speedup w/ ptlu
AES	2.29×
DES	1.28×
3DES	1.25×
RC4	1.92×
Blowfish	1.73×
MARS	1.40×
Twofish	1.61×



(a) ptlu.4.6.2.0 Rd, Rs

(b) ptlu.1.3.0.1 Rd, Rs

Figure 2: Examples of the ptlu **instruction**

- For all ciphers, tables are small and constant in size. Except for RC4, all table accesses are reads.
- Number of entries per table is 256 for 5 of the ciphers (8 index bits), and the data read is either 8 or 32 bits.
- The round structure of the ciphers permit table lookups to be parallelized. For example, all lookups in an AES round (16 lookups) can be performed in parallel, constrained only by hardware resources.

3.2. ISA support for parallel table lookups

The data in Table 5 motivate the use of a dedicated instruction to accelerate the table lookup operations in symmetric-key ciphers. To evaluate its performance benefits, we have added such an instruction, called **ptlu** (*parallel table lookup*), to the PAX architecture, which is a minimalist high-performance cryptographic processor ISA designed at Princeton University [15].

The PAX datapath, which is shown in Figure 1, includes 8 small on-chip tables, denoted T0-T7. Each table has 256 entries and the size of each entry is the same as the processor wordsize. Because the wordsize in the PAX architecture is variable (this is the *wordsize scalability* feature discussed in Sections 3.3 and 4.2), the total size of the tables is 8 kB for 32-bit PAX, 16 kB for 64-bit PAX, and 32 kB for 128-bit PAX. The total table area, which is at most that required by 32 kB of SRAM, is therefore no bigger than today's L1 caches.

The **ptlu** instruction can perform up to 4 simultaneous table lookups and has the following format:

ptlu.*subword.table.offset.step* Rd, Rs

The *subword*, *table*, *offset*, and *step* fields are given as sub-ops in the instruction. The *subword* field selects the size of the data (in bytes) read from the table. The *table* field is 3 bits and is used to select one of the eight tables for lookup. The byte-sized indices used to access the tables are read from the source register *Rs*. The 4-bit *offset* field is used to select the first byte-sized index in *Rs*. The 4-bit *step* field gives the distance (in bytes) between the subsequent bytes in *Rs* that are used as indices when multiple lookups are performed. Because certain combinations of the four sub-op fields will not be meaningful, the programmer or the compiler is responsible for avoiding these.

Figure 2 contains 2 examples of the **ptlu** instruction for a processor with 32-bit wordsize. In Figure 2a, **ptlu** is used for a single lookup from T6, using the third byte of *Rs* as index. This type of lookup is common in symmetric-key algorithms. Figure 2b is an example of the byte substitution operation for a 32-bit word. Four parallel lookups from T3 are made, where the index for each lookup is a different byte of *Rs*. AES key expansion [5] uses this type of byte substitution.

3.3. Performance of ptlu instruction

For the **ptlu** instruction in PAX processors, we assume single-cycle execution latency because each of the lookup tables is only 1 to 4 kB in size (very small, and hence can be very fast). Because there are no effective address computations with the **ptlu** instruction, the data in the tables can be read in the execution stage of the instruction and forwarded to another functional unit in the next cycle. (In a standard **load** instruction, the load-use interlock will normally prohibit such immediate forwarding of the loaded data.) Another benefit of using on-chip tables is the invariability of the access time for any single table lookup. Unlike the data memory, where a single lookup can take either a single cycle (cache hit) or many cycles (cache miss), a **ptlu** access always takes a single cycle.

Table 6 shows the performance gains (speedups) achieved with the **ptlu** instruction. AES, which has the highest dependency on table lookups, also shows the highest speedup of 2.29×. DES/3DES shows only 28% speedup due to its higher reliance on permutations and the structure of its lookup tables that do not permit full parallelization with the **ptlu** instruction. Speedups for the remaining algorithms vary between 1.40× and 1.92×.

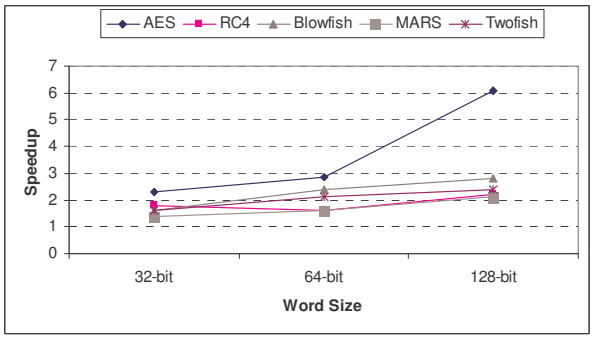


Figure 3: Speedup with ptlu **at different wordsizes**

An important advantage of **ptlu** is its scalability to processors with different wordsizes. Figure 3 shows the performance gains for the AES and AES finalist algorithms when the wordsize is increased from 32 bits to 64 bits to 128 bits. We see that as the wordsize increases, the additional speedup from the **ptlu** instruction increases significantly because more table lookups can be performed simultaneously. For example, there are 16 lookups in a single AES round, which can be completed using only four **ptlu** instructions when the wordsize is 128 bits. Without **ptlu**, these 16 lookups take at least 48 instructions due to the effective address calculation overhead even if all the reads are cache hits. The overall speedups for AES with **ptlu** are 2.29×, 2.85×, and 6.10×, at 32-bit, 64-bit, and 128-bit wordsizes respectively.

3.4. Past work on table lookups

Inclusion of a dedicated instruction to accelerate the table lookup operations in symmetric-key ciphers have previously been proposed in [16] and [17]. For example, the **sbox** instruction in [16] can perform very fast lookups of tables situated in the normal memory space 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, and (2) each table must be aligned to 1024-byte boundaries. Furthermore, cache misses are still possible since the **sbox** instruction uses the normal memory space. In contrast, the **ptlu** instruction can read variable-sized data from a table entry. Furthermore, because dedicated on-chip tables are used, data alignment is not a concern and cache misses are completely avoided.

A slightly different **sbox** instruction is used in the CryptoManiac processor [17], which stores its data on four on-chip 1 kB caches. However, with either **sbox** instruction, only a single table lookup can be performed (per instruction), so multiple-issue techniques such as superscalar execution are needed to achieve multiple parallel table lookups. In contrast, up to 4 table lookups can be done in parallel in a single cycle using a single **ptlu** instruction. For fast symmetric-key ciphers like AES, in order to match the performance of a single-issue processor with **ptlu**, a multi-way superscalar (4 or 8-way) processor is needed with **sbox**. However, such wide multi-issue architectures may not be suitable for mobile devices with limited resources.

4. Public-key and digital signature algorithms

4.1. Workload characteristics

We use the SimpleScalar toolset to profile the public-key algorithms included in mCrypt, which are listed in Table 7. Table 8 shows the breakdown of execution times at the macro-operation level. In order to make a fair comparison between non-ECC and ECC algorithms, we have assumed the availability of a dual-field multiplier that can perform both integer and binary-field multiplication. Without a dual-field multiplier, binary-field multiplication is realized in software, resulting in poor overall ECC performance on programmable processors. It is described in [18] how an ordinary integer multiplier can be converted to a dual-field multiplier with only minor changes.

Despite the differences in key size and the mathematical framework, ECC and non-ECC algorithms exhibit similarities in that the execution time of either type of algorithm is dominated by the multiplication operations on the underlying field.

4.2. Performance impact of wordsize scaling

Table 9 shows how the performance of a public-key algorithm changes as the wordsize of a single-issue processor is scaled from 32 bits to 64 bits to 128 bits. (Since we use a dual-field multiplier whose inputs are wordsized, these numbers are also the size of each multiplier input.) For brevity, we choose to focus only on eDH because it is representative of the other ECC algorithms [19]. Behavior of eDH is also similar to the non-ECC algorithms since multi-precision multiplication is the dominant operation in either case.

We observe that doubling the wordsize to 64 bits more than quadruples the performance, and quadrupling the wordsize to 128 bits gives speedups of 23.32× to 27.67×. To put this huge improvement in perspective, the speedup obtained with an 8-way superscalar 32-bit processor with two multipliers and two memory pipes is only 4.31×.

Table 7: Public-key algorithms in mCrypt

Name	Typical key size (bits)	Based on Hard Problem	Dominant Operation	Description
Diffie-Hellman (DH) [6]	1024	Discrete Logarithm	Integer multiplication	DH is used in TLS/SSL and SSH. ElGamal is used in DSS. DSS is the US federal standard for digital signatures. RSA is used in many security standards/protocols: S/MIME, IPSec, TLS/SSL, S/WAN, PKCS, IEEE P1363, and WAP WTLS.
El-Gamal [6]				
Digital Signature Algorithm (DSA) [4]				
RSA [6]		Integer Factoring		
RSA Signature [6]				
Elliptic-Curve Diffie-Hellman Key Exchange (eDH)[1]	163	Elliptic-Curve Discrete Logarithm	Binary field (polynomial) multiplication	These are the ECC analogs of DH, ElGamal, and DSA respectively. RSA does not have an elliptic-curve counterpart.
Elliptic-Curve El-Gamal (eEl-Gamal)				
Elliptic-Curve Digital Signature Algorithm (eDSA) [4]				

[1] In all ECC algorithms, we use the NIST-specified 163-bit random binary curves [4]. While prime fields can also be used, binary fields with polynomial basis are preferred because this yields the simplest and fastest software implementations.

Table 8: Percentage of execution time consumed by different operations

	Algorithm / %							
Operation	DH	ElGamal	RSA	RSA Sign.	DSA	eDH	eElGamal	eDSA
Integer Mult.	99	98	98	98	94	-	-	-
Binary Mult.	-	-	-	-		99	98	95
Other	1	2	2	2	6	1	2	5

Table 9[1]: Impact of wordsize

	Key Size	Word Size / Speedup		
Algorithm	(bits)	32-bit	64-bit	128-bit
RSA	1024	1.00×	5.33×	23.32×
eDH	163	1.00×	5.56×	27.67×

[1] RSA is representative of other integer-based public-key algorithms; eDH is representative of other ECC algorithms that use binary fields and polynomial basis representation for field elements.

4.3. Multi-word result (MR) execution

Previous section shows that wordsize scaling is a very effective ISA tool in improving public-key encryption performance. Unfortunately, it is not applicable to existing servers, which have fixed ISAs and fixed wordsize. However, some of the benefits of a larger wordsize can be obtained with *multi-word result* (MR) functional units.

Figure 4 depicts the differences between a standard (1-word result, or 1-R) multiplier and a multi-word result multiplier. The 1-R multiplier executes two instructions, **mul.lo** and **mul.hi**, to write either the lower or the higher half of the product to the result bus. With minor modifications to the datapath, a 2-word result (2-R) multiplier is obtained (Figure 4b). Using this, the full 64-bit product of two 32-bit multiplicands can be generated with a single instruction.

We simulate 2-R multiplier execution by dynamically monitoring the instruction issue window and looking for consecutive **mul.lo/mul.hi** pairs using the same source registers. For example:

mul.lo Rd1, Rs1, Rs2; mul.hi Rd2, Rs1, Rs2

When such pairs are detected, the two instructions are issued as a single multiply instruction, and Rd1 and Rd2 get the low and high halves of the product respectively.

Table 10 shows the performance benefits of using a 2-R multiplier for RSA and eDH. These are representative of the other integer and ECC algorithms respectively. The speedups are 1.43× and 1.39× for RSA and eDH respectively, which are lower than the speedups for full wordsize scaling (Table 9). However, MR execution has the advantage that no ISA changes are needed, hence it can be implemented in servers with a fixed wordsize.

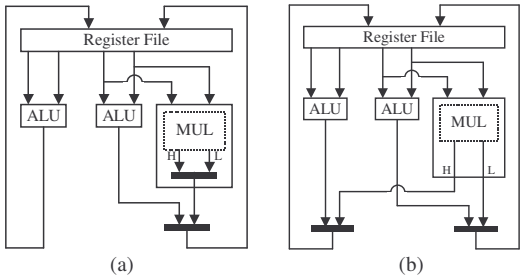


Figure 4: (a) 2-way superscalar datapath (b) Modified datapath for 2-R multiplier execution

Table 10: Speedups with 2-R result execution

		Speedup	
Algorithm	**Key Size** (bits)	2-way superscalar with 1 standard (1-R) multiplier	2-way superscalar with 1 2-R multiplier
RSA	1024	1.00×	1.43×
eDH	163	1.00×	1.39×

5. Performance scaling for wireless

Security protocols use a combination of symmetric-key, hash, and public-key algorithms. For example, the WTLS protocol [9] has a *handshake phase* for authentication (public-key) and a *record phase* for encryption and integrity checking (symmetric-key + hash).

In Figure 5, we show the number of clients that a 1 GHz server can authenticate in 1 second with WTLS

handshake. We consider ECC and RSA based handshakes, with and without the multi-word result functional units. The numbers on the horizontal axis indicate the processor configuration specified as *superscalar issue width / number of memory pipes / number of multipliers*. Since the number of supportable clients scales linearly with the number of servers, the clock rate, and the authentication latency, we can use Figure 5 to derive results corresponding to any combination of these parameters.

Figure 6 shows the clock rate required to complete a WTLS handshake within a given latency. We consider both ECC-based and RSA-based handshakes at three different wordsizes (shown in parenthesis) using standard multipliers. Figure 6 clearly demonstrates the huge impact of wordsize on public-key performance.

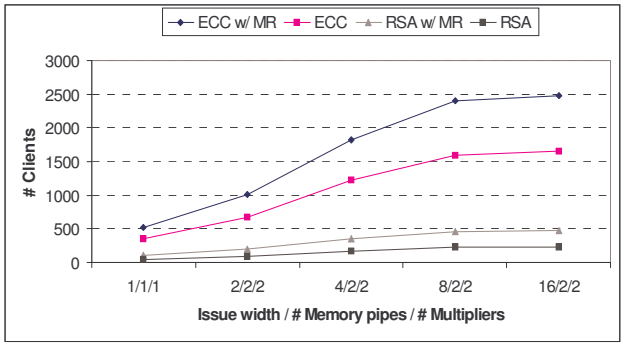


Figure 5: Number of WTLS handshakes without client authentication

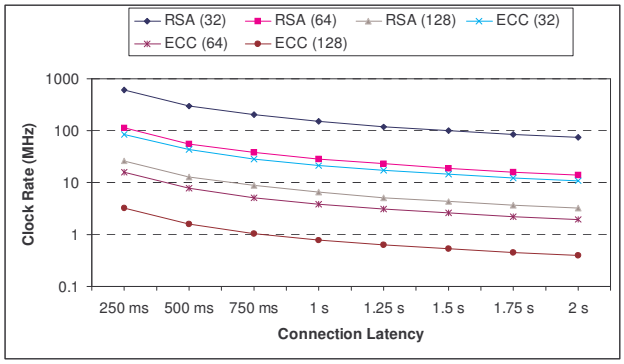


Figure 6: Clock rate for a given handshake delay

For AES, we report in Table 11 the execution cycles required per block of encryption for 4 different processor configurations. The first setup is a 32-bit MIPS-like basic RISC instruction set (used as baseline); the remaining 3 setups are 32-bit, 64-bit, and 128-bit processors extended with the **ptlu** instruction. The numbers in parenthesis indicate the speedup obtained through the use of wordsize scaling and the **ptlu** instruction.

In contrast to AES, which benefits significantly from the **ptlu** instruction and larger wordsizes, the performance of SHA-1 is similar for all four processor configurations, averaging 1200 cycles per 512-bit block (Table 11). This is due to the round structure of SHA-1, which consists of many serial operations on 32-bit words and hence cannot be parallelized as easily as AES in order to efficiently utilize larger wordsizes. Based on the data in Table 11, we compute in Table 12 the combined AES+SHA-1 throughput (as used in WTLS record) for a 400 MHz processor. This clock rate is widely used today by high-end PDAs. Here, the combination of wordsize scaling and the **ptlu** instruction provides speedups between 1.73× (32-bit wordsize) and 2.67× (128-bit wordsize).

Table 11: Execution cycles for AES

	32-bit Basic RISC	**32-bit w/ ptlu**	**64-bit w/ ptlu**	**128-bit w/ ptlu**
AES encryption (per 128-bit block)	878 (1.00×)	384 (2.29×)	308 (2.85×)	144 (6.10×)
SHA-1 hash (per 512-bit block)	~ 1200 cycles			

Table 12: Combined AES+SHA throughput at 400 MHz

	32-bit Basic RISC	**32-bit w/ ptlu**	**64-bit w/ ptlu**	**128-bit w/ ptlu**
AES + SHA-1 (WTLS-record)	41.61 Mbps (1.00×)	71.86 Mbps (1.73×)	80.91 Mbps (1.95×)	111.1 Mbps (2.67×)

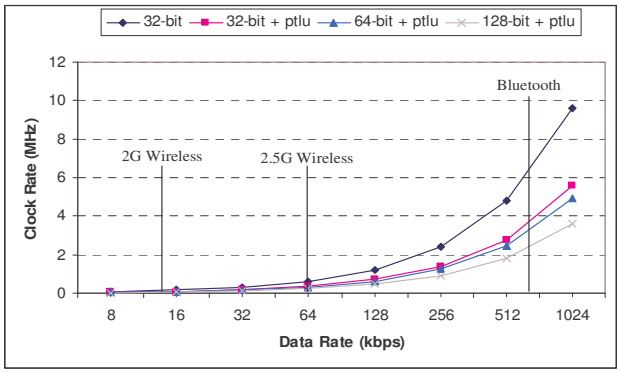


Figure 7: Clock rate required for AES+SHA-1 combined data rates between 8 kbps – 1024 kbps

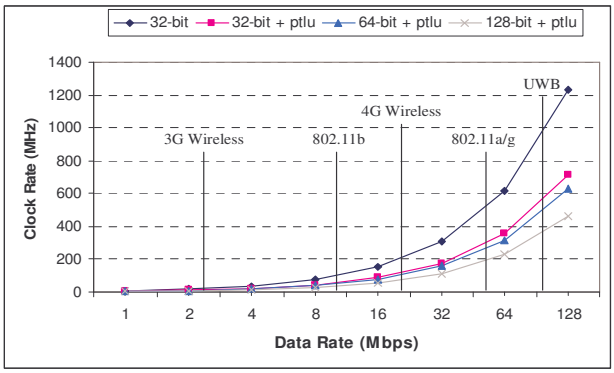


Figure 8: Clock rate required for AES+SHA-1 combined data rates between 1 Mbps – 128 Mbps

Based on the number of cycles taken to execute AES+SHA-1 in Table 11 for each of 4 processors, we can also compute the processor clock rates required to meet link speeds from 8 kbps through 128 Mbps, as shown in Figure 7 and Figure 8. In these figures, we also indicate by vertical lines the link speeds of various wireless technologies such as 2G-4G cellular wireless, Bluetooth, WiFi (802.11b), and UWB [1][8]. For example, to achieve combined AES+SHA-1 processing at the 64 kbps 2.5G cellular wireless link speed, a 0.225 MHz clock is required for a 128-bit processor with the **ptlu** instruction. Similarly, to meet the 54 Mbps link speed of the IEEE 802.11a/g wireless technology [8], the clock rate needs to be 519 MHz for a 32-bit processor without **ptlu**, and only 194 MHZ for a 128-bit processor with **ptlu**.

6. Conclusions

Cryptography algorithms are essential building blocks used to provide security on public communication networks like the Internet. In this paper, we first selected a suite of cryptography algorithms, *mCrypt*, which is representative of the existing major security protocols. Since *mCrypt* includes symmetric-key, hash, and public-key algorithms, our work differs from previous studies [16][17], which only consider symmetric-key ciphers.

Our study of the workload characteristics of symmetric-key ciphers showed that most modern ciphers use table lookups in their round structures, during which they spend a major fraction of their aggregate execution time (72% for AES). To accelerate these table lookups, we described the **ptlu** instruction, which we have included in the PAX cryptographic processor ISA. This gave us a 2.29× speedup (for AES), which further increased to 6.10× when **ptlu** was used together with wordsize scaling.

We then studied the public-key ciphers and verified that wordsize is a major determinant of performance (e.g. 27.67× speedup for 4× wordsize scaling). The impact of wordsize scaling is far higher than the speedups obtained with multiple-issue execution (e.g. 4.31× speedup with 8-way superscalar). For servers where wordsize scaling is not practical, we showed how *multi-word result* functional units can be used. This provides speedups of about 40% without necessitating any ISA changes.

Finally, we computed the parameters required by a server to simultaneously authenticate a growing number of clients within a given latency. For clients, we computed the parameters required for the encryption+hashing performance to meet the links speeds of existing and emerging wireless technologies.

A major contribution of this paper is the demonstration of the effectiveness of wordsize scaling as a technique for significantly improving performance for both symmetric-key and public-key cryptographic processing. For such workloads, the speedup obtained with wordsize scaling is far higher than increasing the number of instructions executed per cycle ("IPC scaling") in superscalar or VLIW execution, with lower implementation complexity.

References

[1] T.S. Rappaport et al., "Wireless Communications: Past Events and A Future Perspective," *IEEE Communications Magazine,* vol. 40, no. 5, pp. 148-161, May 2002.

[2] V.S. Miller, "Use of Elliptic Curves in Cryptography", *Lecture Notes in Computer Science*, vol. 218, Springer-Verlag, pp. 417-426. 1986.

[3] N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, vol. 48, no. 177, pp. 203-209, 1987.

[4] NIST, Digital Signature Standard (DSS), FIPS Pub. 186-2, <http://csrc.nist.gov/publications/fips>, Feb. 2000.

[5] NIST, Advanced Encryption Standard (AES), FIPS Pub. 197, <http://csrc.nist.gov/publications/fips>, Nov. 2001.

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

[7] A.K. Lenstra and E.R. Verheul, "Selecting Cryptographic Key Sizes", *Journal of Cryptology*, vol. 14, no. 4, pp. 255-293, Dec. 2001.

[8] IEEE 802.11 Working Group, IEEE 802.11 Wireless LAN *Standards,* <http://grouper.ieee.org/>.

[9] Wireless Application Protocol 2.0 – Technical White Paper, WAP Forum, <http://www.wapforum.org>, Jan. 2002.

[10] Submissions to the 1st AES Conference, Aug. 1998.

[11] NIST, Secure Hash Standard (SHS), FIPS Pub. 180-2, <http://csrc.nist.gov/publications/fips>, Aug. 2002.

[12] W. Stallings, Cryptography and Network Security: Principles and Practice, Prentice Hall, 1998.

[13] 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*.

[14] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0.", *Computer Architecture News*, pp. 13-25, June 1997.

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

[16] 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.

[17] 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, 2001.

[18] J. Großschädl and G.-A. Kamendje, "Instruction Set Extension for Fast Elliptic Curve Cryptography Over Binary Finite Fields GF(2^m)", *Proc. IEEE Int. Conf. Application-Specific Systems, Architectures, and Processors (ASAP)*, pp. 455-468, Jun. 2003.

[19] A.M. Fiskiran and R.B. Lee, "Workload Characterization of Elliptic Curve Cryptography and other Network Security Algorithms for Constrained Environments", *Proc. IEEE Int. Workshop on Workload Characterization (WWC)*, pp. 127-137, Nov. 2002.