Encryption Overhead in Embedded Systems and Sensor Network Nodes: Modeling and Analysis

Ramnath Venugopalan, Prasanth Ganesan, Pushkin Peddabachagari,
Alexander Dean, Frank Mueller, Mihail Sichitiu

Center for Embedded Systems Research, Depts. of ECE and CS
North Carolina State University, Raleigh, NC 27695

{agdean, mlsichit}@unity.ncsu.edu, mueller@cs.ncsu.edu, phone: (919) 515-7889, fax: -7925

ABSTRACT

Recent research in sensor networks has raised issues of security for small embedded devices. Security concerns are motivated by the deployment of a large number of sensory devices in the field. Limitations in processing power, battery life, communication bandwidth and memory constrain the applicability of existing cryptography standards for small embedded devices. A mismatch between wide arithmetic for security (32 bit word operations) and embedded data bus widths (often only 8 or 16 bits) combined with lack of certain operations (e.g., multiply) in the ISA present other challenges.

This paper offers two contributions. First, a survey investigating the computational requirements for e a number of common cryptographic algorithms and embedded architectures is presented. The objective of this work is to cover a wide class of commonly used encryption algorithms and to determine the impact of embedded architectures on their performance. This will help designers predict a system's performance for cryptographic tasks. Second, methods to derive the computational overhead of embedded architectures in general for encryption algorithms are developed. This allows one to project computational limitations and determine the threshold of feasible encryption schemes under a set of the constraints for an embedded architecture.

Experimental measurements indicate uniform cryptographic cost for each encryption class and each architecture class and negligible impact of caches. RC4 is shown to outperform RC5 for the Atmega platform. But when message authentication is required in addition to encryption, hash or block ciphers, such as RC5, have the advantage of providing support for both authentication and encryption. The analytical model allows to assess the impact of arbitrary embedded architectures as a multi-variant function for each encryption scheme. Overall, our results are not only valuable to assess the feasibility of encryption schemes for existing embedded architectures, they also extend to assess the feasibility of encryption methods for new algorithms and architectures for sensor systems.

CASES'03, Oct. 30 – N. 2, 2003, San Jose, California, USA.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems.

E.3 [**DATA ENCRYPTION**]: Standards.

General Terms

Measurement.

Keywords

Embedded Systems, Security, Encryption, Sensor Networks.

1. Introduction

Security is a well-established field for general-purpose computing. Security mechanisms address computing services, such as authentication for user admission, intrusion detection and prevention as well as counter-measures for other forms of attacks (e.g., denial of service) and data protection in storage, in e-mails or to provide secure transactions. This paper focuses on the last aspect, namely, data protection mechanisms provided by encryption techniques. The objective of this paper is to study the impact of a variety of encryption techniques for *embedded* architectures instead of general-purpose processors.

Embedded systems have a long history in the context of transaction processing, for example, cash transactions at teller machines. However, security measures have typically focused on physical access restrictions as well as software measures to disable a device if attempts to tamper with it are suspected. Recent developments have changed this focus. On the one side, embedded architectures provide a wider range of processing power, which allows more

sophisticated security responses, in particular for high-end embedded systems. On the other side, new application areas in embedded systems require secure communication. For example, recent work in sensor networks includes data encryption considerations [12]. Sensor networks allow the collection of data from low-end sensor nodes in the field. This data is communicated over non-secure channels, such as radio frequencies, through routers (in the latest design) and, ultimately, to a base station for further processing and decision making. Applications range from battlefield surveillance over data collection to study environmental impacts to medical observation. Beyond sensor networks, embedded processors are increasingly deployed with network connections, such as in PDAs with wireless communication (802.11b)[23], e.g., for the Ipaq Pocket PCs used in this study [1]. The objective of data encryption in such settings is to ensure that data can only be interpreted by authorized recipients.

In this paper, we assess the feasibility of different encryption schemes for a range of embedded architectures. We determine architectural impacts on the performance of encryption as well as algorithmic properties of the selected encryption schemes. The particular embedded platforms were chosen to cover a wide range of embedded devices. Measurements were obtained for six different architectures, ranging in word size from 8 (Atmel AVR) over 16 (Mitsubishi M16C) to 32-bit width (StrongARM, XScale) to cover low-end, medium and high-end embedded processors, respectively. As a baseline for comparison, one general-purpose architecture (SPARC) was also included as a reference point. Future encryption schemes need only be evaluated on reference architectures to derive the overhead for other architectures. Other reference architectures are those with differing ISA support for encryption, as detailed in the evaluation. The analysis takes into account features of architectures, such as processor frequency, ISA characteristics, such as RISC vs. CISC, support for variable-sized bit shifts or native multiply, and the impact of memory hierarchies for architectures with caches.

Five common encryption schemes were chosen for the study ranging from stream ciphers (RC4) over block ciphers (RC5, IDEA) to hashing techniques (SHA-1, MD5). This choice was driven by the objective to assess encryption schemes with different overheads that provide increasing levels of protection. Most significantly, the algorithmic choice is motivated by the constraints of embedded architectures. Public key encryption schemes do not appear to be feasible on current low-end embedded systems, not only because of code/data size and processing constraints but also due to their high demand on power consumption, which would severely limit the lifetime of mobile devices such as nodes in a sensor network.

We obtained measurements to assess the overhead of encryption for the aforementioned algorithms and platforms. We studied the impact of the length of the data to encrypt as well as a variety of processor-dependent parameters, as mentioned above. Results indicate a mostly uniform cycle overhead for each word size (8/16/32 bit) but differences between the three word-size classes. The impact on caches was negligible while ISA support is limited to specific effects on certain algorithms. Specifically, we were surprised to find that RC4 outperforms RC5 on encryption for the 8 and 16-bit architectures. This is particularly interesting since RC5 was chosen for the Atmega in the Berkeley Motes SPINS project [12]. Although the choice of RC5 for SPINS was due to memory constraints and message authentication – the block cipher could also be used as a hash function – other 8-bit architectures may fare better with RC4, as our results show. However, if message authentication is required, a hash or block cipher would be needed in addition to RC4, which makes other schemes, such as RC5, more efficient due to their versatility. We also found that hashing techniques require almost an order of a magnitude higher overhead. Based on our results, we formulate an analytical model to assess the impact of arbitrary embedded architectures as a multi-variant function for each encryption scheme depending on processor frequency, word width, ISA type and specific ISA support.

The paper is structured as follows. First, we contrast the different encryption schemes and embedded platforms. Following this survey, we present and interpret measurements from a variety of experiments. From these results, we derive an analytical model. Our discussion of related as well as future work and a summary of our contributions conclude the paper.

2. Algorithms

Our choice of algorithms represents common symmetric encryption and hashing function schemes that form an integral part of many security protocols. *RC4* [2] is used in IEEE 802.11 WEP [13], *IDEA* [2] and *MD5* [2],[3] are part of PGP [11], *SHA-1* [4] and *MD5* [2][3] are included in the security architecture for Internet Protocol (IPSEC) [14],[10], and RC5 [1] has been suggested as a good algorithm for sensor networks [12]. These algorithms offer variety in the mode in which they operate and encompass different mathematical and data manipulation operations. They work on different word sizes ranging from 8 bits to 32 bits, and, hence, help assess the effectiveness of the different architectures. Table 1 presents the parameters used in our study.

Table 1: Encryption Schemes and Parameters

Algorithm	Type	|key/hash|	|Block|
RC4 [2]	stream	128 bits	8 bits
IDEA [2]	block	128 bits	64 bits
RC5 [1]	block	64 bits	64 bits
MD5 [2][3]	1-way hash	128 bits	512 bits
SHA1 [4]	1-way hash	128 bits	512 bits

RC4 is a stream cipher symmetric key algorithm. This algorithm is quite simple and operations involve the addition of 8 bit elements or swapping variables in a 256-byte state table. RC4 supports variable length keys. We consider a 128-bit key here.

IDEA (International Data Encryption Algorithm) is a symmetric-key block cipher that operates on 64 bit plaintext blocks. The key is 128 bits long with the same algorithm used for both encryption and decryption. The algorithm primarily includes operations from three algebraic groups: XOR, addition modulo 2^{16}, multiplication modulo $2^{16}+1$.

RC5 is a fast symmetric block cipher with a variety of parameters: block size, key size and number of rounds. We currently focus on a RC5 implementation with a 64-bit data block and 64-bit key. It uses the XOR, addition and rotation operations.

MD5 is a one-way hash function that processes the input text in 512 bit blocks to generate a 128-bit hash value. The mathematical operations that are involved in this algorithm are: XOR, AND, OR, NOT and rotations. The algorithm also pads plaintext to 512 blocks with the last 64 bits of the last block indicating the length of the message.

SHA-1 is also a one-way hash function that produces a 160-bit output when any message of any length less than 2^{64} bits is input. The operations are similar to MD5 and constitute XOR, AND, OR, NOT and rotations.

3. Hardware Platforms

We evaluate the performance of the cryptographic functions on five different embedded processors, which were selected to span a broad range of applications from low-end (4 MHz 8-bit Atmel AVR Atmega 103) to high-end (400 MHz 32-bit Intel XScale). For comparison we also evaluate the performance of a workstation (with a 440 MHz 64-bit SPARC CPU, operated in 32-bit mode), as depicted in Table 2.

Table 2: Hardware Platforms

Platform	Wordsize	clockfreq.	I/D-$
Atmega 103	8 bits	4 MHz	none
Atmega 128	8 bits	16 MHz	none
M16C/10	16 bits	16 MHz	none
SA-1110	32 bits	206 MHz	16/8KB
PXA250	32 bits	400 MHz	32/32KB
UltraSparc2	64/32 bits	440 MHz	16/16KB

3.1 Atmega 103/Atmega 128

The Atmega 103 implements the AVR architecture, a RISC architecture featuring 8 bit native word size, 32 general-purpose registers, and limited support for 16 bit operations. The processor features a two-stage pipeline. This processor lacks multiply and divide instructions. Data memory is byte-accessible and byte-aligned. The Atmega 103 is in the low end of the performance spectrum of the AVR device family. We use an Atmel STK300 evaluation board with a 4 MHz clock. On-chip memory consists of 4 kilobytes of SRAM and 128 kilobytes of Flash EEPROM. In addition, 32 kilobytes of external SRAM are used (with a one cycle performance penalty). No cache exists, and no coprocessor is available. The C compiler used is GCC 3.0. No operating system is used.

Running at 16MHz, the Atmega 128 is pin-compatible with the Atmega 103 (which runs only at 4 MHz). With its improved clock rate, the Atmega 128 is at the high end of the AVR family's performance spectrum. The performance is identical on a cycle-by-cycle basis, with the exception of the addition of a two-cycle multiply instruction. Some algorithms use multiplication; these were recompiled for the Atmega 128 and run to derive new execution times. Algorithms that do not use multiplication have identical code whether for the Atmega 103 and 128, and they result in identical cycle counts.

3.2 M16C/10

The Mitsubishi M30102 implements the company's M16C ISA, a CISC architecture featuring a 16 bit native word size, four general purpose registers and six address and pointer registers. This is a widely used architecture in the automotive industry and has been available for over ten years. The CPU is not pipelined; the manufacturer states 75% of instructions take five or fewer cycles to execute. The 16 MHz M30102 is in the middle of the performance spectrum of the M16C device family; other devices are available with clock rates of 24 MHz. There is no coprocessor available. We use a Mitsubishi MSV30102-SKP evaluation board with a 16 MHz M30102 and no external memory. No operating system is used.

This MCU offers 1 KB of SRAM and 24 KB of Flash EEPROM on-board. No cache exists, and the memory is word-aligned (with a one-cycle penalty for misaligned accesses). The C compiler used is Mitsubishi's nc30 version 3.00.01, and -03 optimization is selected.

3.3 StrongARM SA-1110

The SA-1110 is a 32-bit Intel StrongARM RISC processor capable of running at up to 206 MHz that implements the ARM v4 architecture. The SA-1110 MMUs provide separate 32-entry translation look-aside buffers (TLBs) for the instruction and data streams. The SA-1110 contains 16 Kbytes of instruction cache and 8 KB of data cache. The memory bus interfaces to many device types including DRAM, SDRAM and ROM. This processor forms the core of the iPAQ Pocket PC, which was the platform we used to perform the measurements. The Pocket PC comes with 32

MB of RAM. The operating system used was Familiar Linux with code compiled using the GNU gcc compiler.

3.4 XScale PXA250

The PXA250 is a low-power high-performance 32-bit Intel XScale™ core-based CPU (200, 300 and 400 MHz). It is ARM architecture v.5TE compliant and a successor to the StrongARM processor. It is based on Intel's superpipelined RISC technology. The PXA250 has 32 KB of instruction and data caches. This processor is used in iPAQ 39xx series of Pocket PC 2002 with a RAM of 64 MB and 48 MB flash ROM. The iPAQ used in our experiments is powered by Win CE. The eVC++ compiler provided by Microsoft was used for generating code.

3.5 UltraSPARC II

The UltraSPARC II series of microprocessors are 64 bit RISC based architectures. They implement the SPARC v9 architecture. It is a superscalar, superpipelined micro-architecture. It has an on chip instruction cache of 16 KB and on chip data cache of 16 KB. The SPARC processor we used has a frequency of 440 MHz. The processor has an external cache of 2 MB. The SPARC, unlike the embedded architectures, is a generic processor. The Operating System used was Sun Solaris with the code compiled in 32-bit mode using GNU gcc.

4. Experiments & Analysis

In this section, we present the results of the execution times measurements of the considered algorithms on the various microcontroller architectures. We also develop an approximate model for the execution times applicable to any microcontroller architecture.

4.1 Experimental Methods

Experiments were conducted for each architecture and algorithm. For each of the considered platforms, we compiled the same implementation of the considered algorithms without any modifications. Input lengths were varied for encryption based on hashing with fixed-sized packets to assess the effect of algorithmic padding up to packet length. The block and hash algorithms operate on plaintext that meets specific byte boundaries. In case the plaintext is not a multiple of the block size, the plaintext is padded. The RC5 and IDEA implementations work on block sizes of 64 bits. The MD5 and SHA-1 algorithms work on 512 bit blocks. The plaintext that is input to all the symmetric cryptography algorithms is 128 bits long. We work with incrementing sizes of plaintext with the hash algorithms until we approach the second 512-bit block boundary. For one architecture, the XScale, the experiments were conducted for two frequency settings, namely 200 MHz and 400 MHz, while memory access times remained the same. This experiment was conducted to assess the impact of caches on the algorithms, which can be inferred since memory fetches on a miss take fewer cycles for lower processor frequencies while memory latency remains constant. Each functional block of the algorithm, such as initialization, encryption and decryption, was executed 1000 times with the same input, and results were averaged over these runs. The timing information is obtained as system time on all platforms, except for the low-end micro-controllers where built-in timers are used.

4.2 Performance Assessment

Figure 1 depicts the execution time overhead for each of the considered platforms and algorithms on a log scale. These numbers are also depicted in Table 3.

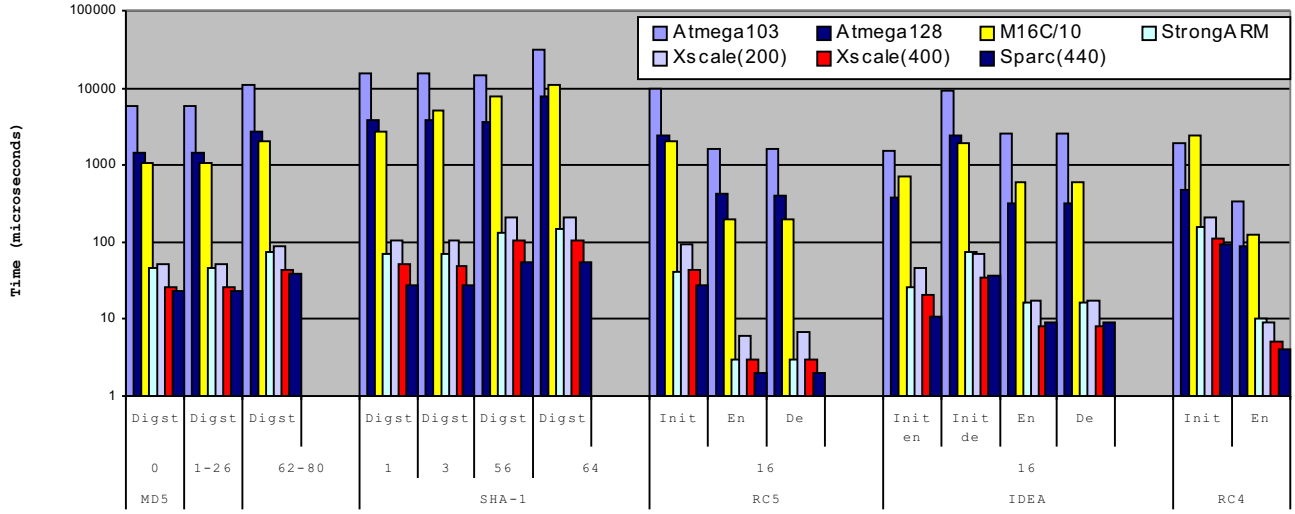

Figure 1: **Execution times [μs] for algorithms, platforms and plaintext sizes [bytes]**

Table 4: Execution times [μs] for algorithms, platforms and plaintext sizes [bytes]

Algorithm	Size	Action	Atmega103	Atmega128	M16C/10	StrongARM	Xscale(400)	Xscale(200)	Sparc(440)
MD5	0	Digest	5863	1466	1083	46	26	53	23
	1-26	Digest	5890	1473	1075	46	26	53	23
	62-80	Digest	10888	2722	2011	74	45	90	39
SHA-1	1	Digest	15249	3812	2651	69	51	102	27
	3	Digest	15781	3945	5303	69	50	103	27
	56	Digest	14543	3636	7955	133	102	205	55
	64	Digest	31107	7777	10907	145	103	207	56
RC5	16	Init	9641	2410	2074	41	45	91	28
		Enc	1651	413	197	3	3	6	2
		Dec	1636	409	202	3	3	7	2
IDEA	16	Init enc	1523	381	727	26	21	47	11
		Init dec	9417	2354	1927	76	35	69	36
		Enc	2555	325	596	16	8	17	9
		Dec	2614	325	597	16	8	17	9
RC4		Init	1886	472	2455	155	108	216	96
		Enc	344	86	123	10	5	9	4

For the digest algorithms (MD5 and SHA1), we used multiple plaintext sizes to emphasize the non-linear behavior of those algorithms with the length of the plaintext. The main reason for this nonlinear behavior is the existence of a minimum plaintext size (64 bytes) for those algorithms, so smaller messages are padded up to the minimum plaintext size. As expected, the slowest microcontroller (Atmega 103 –4 MHz), also the simplest (from the point of view of resources and capabilities), will take the longest time to complete any of the analyzed cryptography algorithms.

A comparison of RC5 and RC4 on AtA comparison of RC5 and RC4 on Atmega 103 reveals that the encrypt times are close to each other. In fact, RC4 is slightly faster. However, a similar comparison on StrongARM indicates RC5 is three times faster than RC4. This can be attributed to the fact that RC5 operates on 32-bit words while RC4 operates on 8-bit words. Since the StrongARM utilizes a 32-bit word size, a 32-bit operation occurs for every 8 bits needed by RC4, thereby reducing the efficiency of the algorithm on higher end architectures. Since RC4 requires accesses to the 256-byte state table for encryption of each byte, the memory access delay can result in larger execution times, but this penalty is almost absent in low-end processors like the Atmega 103.

A comparison between RC5 and IDEA on the Atmega 103 reveals that RC5 is 1.5 times faster than IDEA, although they both work on 64-bit blocks. The workhorse of the IDEA algorithm is the multiply instruction while for RC5 it is rotations. Although both are costly operations on Atmega 103 (since there is a lack of native multiply and variable-length bit shifts), the frequency of the operations makes IDEA more costly.

To isolate the influence of the existence of a multiply instruction we compiled the IDEA algorithm for Atmega 128. Atmega 103 and Atmega 128 microcontrollers almost have identical architectures. The main difference is that Atmega 128 has a native two cycles multiply instruction. Confirming our expectations, the Atmega 128 performs significantly better on IDEA (10220 clock cycles for Atmega 103 vs. 5200 for Atmega 128), i.e., the performance is comparable to the level of RC5.

To eliminate the influence of the clock frequency (which spans two orders of magnitude from 4 MHz for Atmega 103 to 440 MHz for the SPARC), Figure 2 depicts the results in terms of clock cycles instead of wall-clock time, as in Figure 1. Clock cycles, depicted on a logarithmic scale, indicate the overhead in terms of executed instructions for scalar architectures. The most significant observation is that, depending on the word size of the architecture, cycle overhead falls into three classes. Again, consider the impact of the log-scale, which causes diverging results to appear closer than they are.

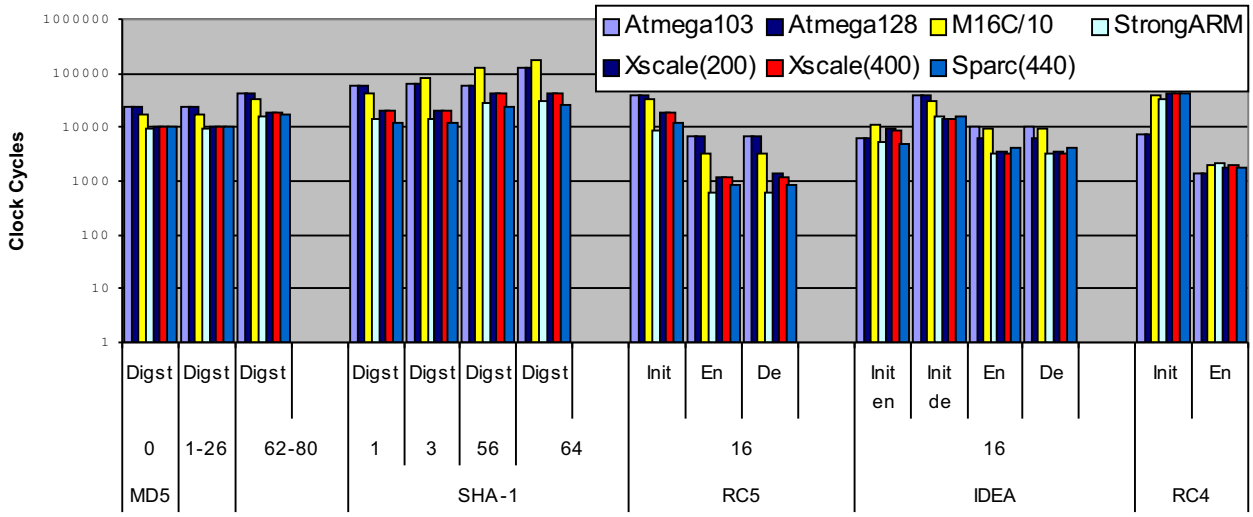

Figure 1: **Clock cycles for algorithms, platforms and various plaintext sizes [bytes]**

Class one, the 8-bit architectures, requires additional loops for architectural shortcomings, such as a missing variable-length bit-shift operation. Instead, the Atmega has to resort to a sequence of single bit-shifts. Class two, the 16-bit architectures, lie between the 8-bit and 32-bit neighbors, as expected. Class three, the 32-bit architectures, comprises a third range of cycle overheads fairly close to each other (StrongARM, XScale and SPARC).

In some cases, the results for the Atmega 103 and the M16C/10 are surprisingly close, which can be attributed to multi-cycle instructions on M16C/10, while the Atmega 128, a RISC with multiply support, performs significantly better. This shows that a RISC design can compensate for its limited instruction set and bus width. The SA1110 and the XScale exhibit similar performance, which stems from their common RISC based ISA at identical bus widths. Both these processors outperform 8-bit and 16-bit micro-controllers roughly by a factor of two. Finally, the SPARC processor, outperformed all other processors in most cases, both in absolute time as well as in clock cycles. This performance of the SPARC is due to a combination of its instruction parallelism (super-scalar RISC design) and multi-level cache hierarchy. Recall that SPARC executables were compiled for the 32-bit SPARC binary format, which means that the SPARC should be treated as a 32-bit architecture in these experiments since its 64-bit design is not being exploited. Notice that the XScale performed slightly better than the SPARC for SHA-1 and IDEA encodes/decodes, which can be attributes to the XScale's larger L1 caches (without L2) and faster memory. Overall, the impact of caches is small. This is realistic given that communicated data will be cached before or after communication for pre- or post-processing, respectively. Hence, messages in excess of 80 bytes should not result in significant changes.

Comparing the two message digest algorithms (MD5 and SHA1), we show that prior results [8,9] extend to embedded architectures: MD5 is significantly faster than SHA1. Similarly, the symmetric key encryption of RC5 outperforms IDEA. The initialization overheads are significant for all encryption algorithms (RC5, IDEA and RC4), especially for small plaintexts (as previously reported for general-purpose ISAs in [11]).

From these results, clear factors emerge in terms of the effect of word size and architecture, memory access latency, costliness of basic operations (such as multiply and rotations) on the overall performance.

We also studied the variance of execution times, which is of particular interest for real-time systems. Results indicate that variances in execution times rarely occur for most encryption algorithms since data processing proceeds without case distinctions, and data accesses tend to be regular as well as pre-cached at encryption time. Few exceptions exist, which are caused by data dependencies, but even then alternate paths tend to be balanced. In general, the algorithms do not contain significant differences in execution due to conditionals, nor do they vary depending on the input length since data padding up to packet size is applied. Hence, our results are not only valuable to assess the feasibility of encryption schemes for arbitrary embedded architectures, they also impact the analysis of worst-case execution times suitable for schedulability analysis in the context of real-time systems.

4.3 Impact of Native Data Size

After normalizing the different clock frequencies in Figure 2, we still observe a significant performance gap between different classes of processors. More precisely, architectures with larger word size perform better than architectures with smaller words. This is expected because most cryptographic methods use operations on large words.

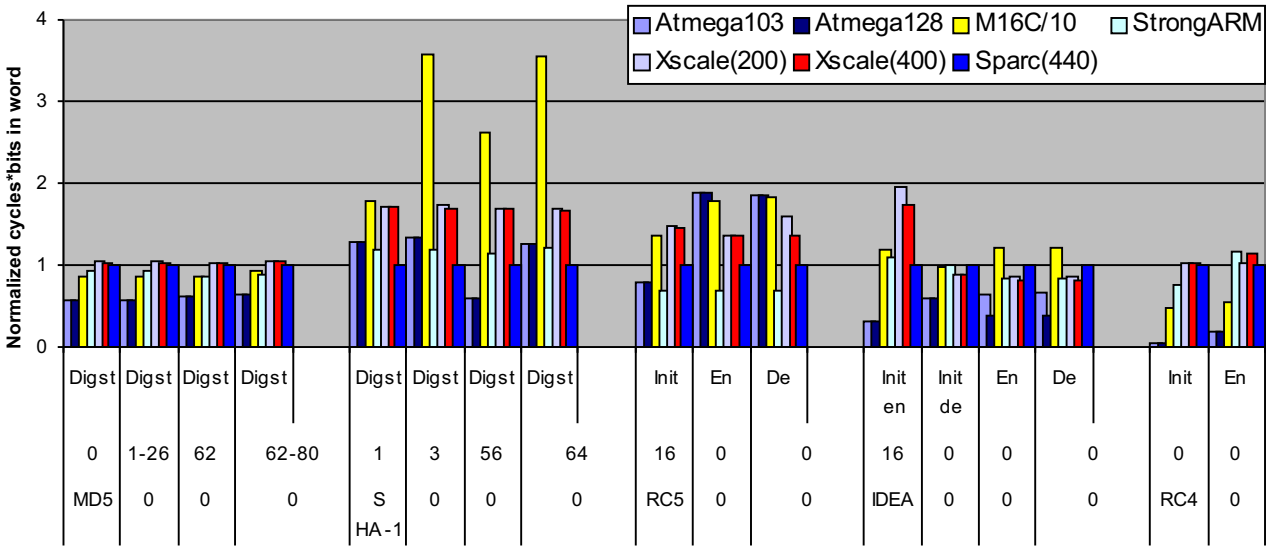

Figure 2: Normalized overhead for algorithms, platforms and plaintext sizes [bytes]

Naturally, implementing large bit operations on architectures with large bus widths is more efficient than implementations on those with a small bus. For a meaningful comparison of different architectures, we consider the influence of various bus sizes. Figure 3 shows the time measurements normalized both as a function of the clock frequency and of the bus width, and then compared with (divided by) the SPARC processor performance. The lower a bar is, the more efficient its ISA and the better it is able to use its native word width. Bars below 1 are possible due to other ISA factors which improve efficiency relative to the baseline architecture, such as single-cycle multibit shifting and fast memory access.

The results in Figure 3 show that the performance overhead normalized by the word width and relative to the reference architecture (SPARC) is surprisingly close for most algorithms and platforms. By normalizing by the word size, we introduce a novel metric that provides a refreshing view from a different angle. The surprisingly close results were somewhat unexpected given the significant differences not accounted for in the normalization operation (number of registers, availability of certain instructions in the ISA, presence and size of cache memory, RISC/CISC architecture etc.). Hence, we conclude that, on the average, these variables do not influence the execution times significantly. The only outlier is SHA-1 for the M16C.

Figure 3 also depicts a few outliers. The M16C performs poorly on our metric for SHA-1, which indicates architectural problems with data sizes and operations of the algorithm. Furthermore, the Atmega microcontrollers are leading the pack with the lowest normalized performance overhead for some algorithms (MD5, IDEA, RC4).

This can be explained by the fact that only *some* operations benefit from larger bus sizes while others, such as branch operations, do not. In addition, given the low clock speed of the Atmega processor, there is no need to resort to methods which raise clock speed at the expense of lowering instructions-per-cycle, as with the StrongARM, Xscale and Sparc. Overall, the Atmega has an efficient RISC architecture that fares well considering its small bus width. Hence, narrow bus low-end processors may provide higher throughput per clock cycle at a low clock rate while high-end architectures provide higher throughput per time unit at high clock rates.

4.4 Code Memory Size

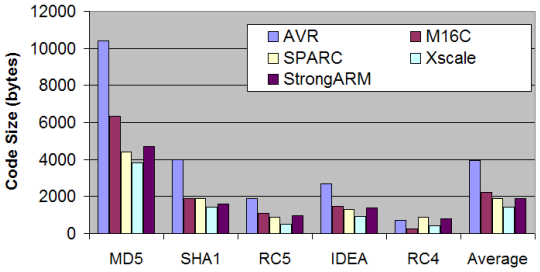

Figure 4: Code sizes for algorithms

Sensor nodes may be implemented with low-cost processors which lack large amounts of program memory, making code size important. Figure 4 shows the code sizes for the cryptographic functions but excludes all scaffolding, library and other code. MD5 requires significantly more code than other algorithms, while RC4 is the most efficient. The AVR architecture requires significantly more space than other ISAs for its code due to its limited instruction set

and eight-bit native data. Finally, the StrongARM requires much more memory than the other 32 bit architectures, which appears to be due to the development tools.

4.5 Performance Model

We observed that the word length and architectural features, namely the complexity of the ISA (RISC vs. CISC) and support for certain ALU operations (variable-sized shifts, multiply) are the causes of variations. From these findings and the experimental data, we can derive a multi-variant model that allows the interpolation of performance for other architectures. The objectives of such a model are threefold. First, the feasibility of existing encryption schemes can be derived by just implementing one scheme on an architecture. Second, encryption overhead can be assessed based on architectural parameters to drive architecture design for a specific encryption scheme and formulate minimum requirements. Third, new encryption schemes only need to be assessed on a subset of reference platforms while their performance on other platforms can be derived from the model.

First, a simple model is introduced. The results of this model are imprecise as there are many variables that influence the execution times of any program (e.g., the presence of variable-sized bit shift and multiplication instructions, presence and size of cache memory, RISC vs. CISC design etc.). The objective of this model is to aid a designer in computing a rough estimate of the execution times for a given encryption algorithm and a particular microprocessor. This rough estimate is especially useful for new architectures. It will allows one to assess if a certain encryption (or hashing) will meet given timing constraints for this particular algorithm, on a projected architecture. Hence, the objective is provide approximate (accurate to a factor of two) execution times of the algorithms. We derived the following performance model:

$$t_{exec}(txt\_len) \approx \frac{a + b \bullet \lceil text\_length / blocksize \rceil}{processor\_freq \bullet bus\_width} \quad (1)$$

where text_length is the size of the plaintext in bytes, processor_frequency and bus_width are the frequency and bus width of the microcontroller, respectively. The parameters **a** and **b** depend on the algorithm being evaluated, and block_size is the size of the blocks in the algorithm. Parameter **a** includes all the initialization overheads while **b** captures the time spent in operations repeated for each block.

For the algorithms considered, we derive the parameters **a** and **b**, which minimize the least square relative error as given in Table 4. Checking the model against the measured results, one can see that most values are within 10%--20% of the measured value. For some measurements and architectures the error is almost twice (or half) of the measured value. This motivates the need to refine the model, as discussed in the following.

Table 6: Parameters for performance model

Algorithm	A	B	blksize(bits)
MD5	203656	86298	512
SHA1	60980	458660	512
RC5 init/encrypt	352114	40061	64
RC5 init/decrypt	352114	39981	64
IDEA encrypt	67751	80617	64
IDEA decrypt	385562	84066	64
RC4	68540	13591	8

The model in (1) is refined to account for other parameters that affect the execution times. For example, some algorithms can take advantage of the existence of a multiply instruction. In Figure 3, it becomes evident that the architecture of the microprocessor (RISC vs CISC) favors the short instructions of the RISC architecture.

Therefore, a more detailed model for the parameters a and b can be derived as follows:

$$a = a_{BASE} + a_{MUL} + a_{RISC} \quad (2)$$

$$b = b_{BASE} + b_{MUL} + b_{RISC} \quad (3)$$

where a_{BASE} and b_{BASE} are the base parameters shown in Table 4, a_{MUL} and b_{MUL} are adjustments of those parameters, which take into account the presence of absence of a multiplication instruction, and a_{RISC} and b_{RISC} take into account the type of the microprocessor architecture (CISC/RISC). For algorithms not using multiplication (e.g., MD5), the adjustments a_{MUL} and b_{MUL} will be zero. For algorithms that can take advantage of a multiplication operation (e.g., IDEA) the parameters a_{MUL} and b_{MUL} can be computed by comparing the results for Atmega 103, which does not have the multiplication instruction, and the other microcontrollers. The adjustments a_{MUL} and b_{MUL} resulting from this comparison for the IDEA encryption algorithm are:

Table 5: Parameters a_{MUL} and b_{MUL} for IDEA

	a_{MUL}	b_{MUL}
w/ MUL instr.	19016	-1143
w/o MUL instr.	-14330	8252

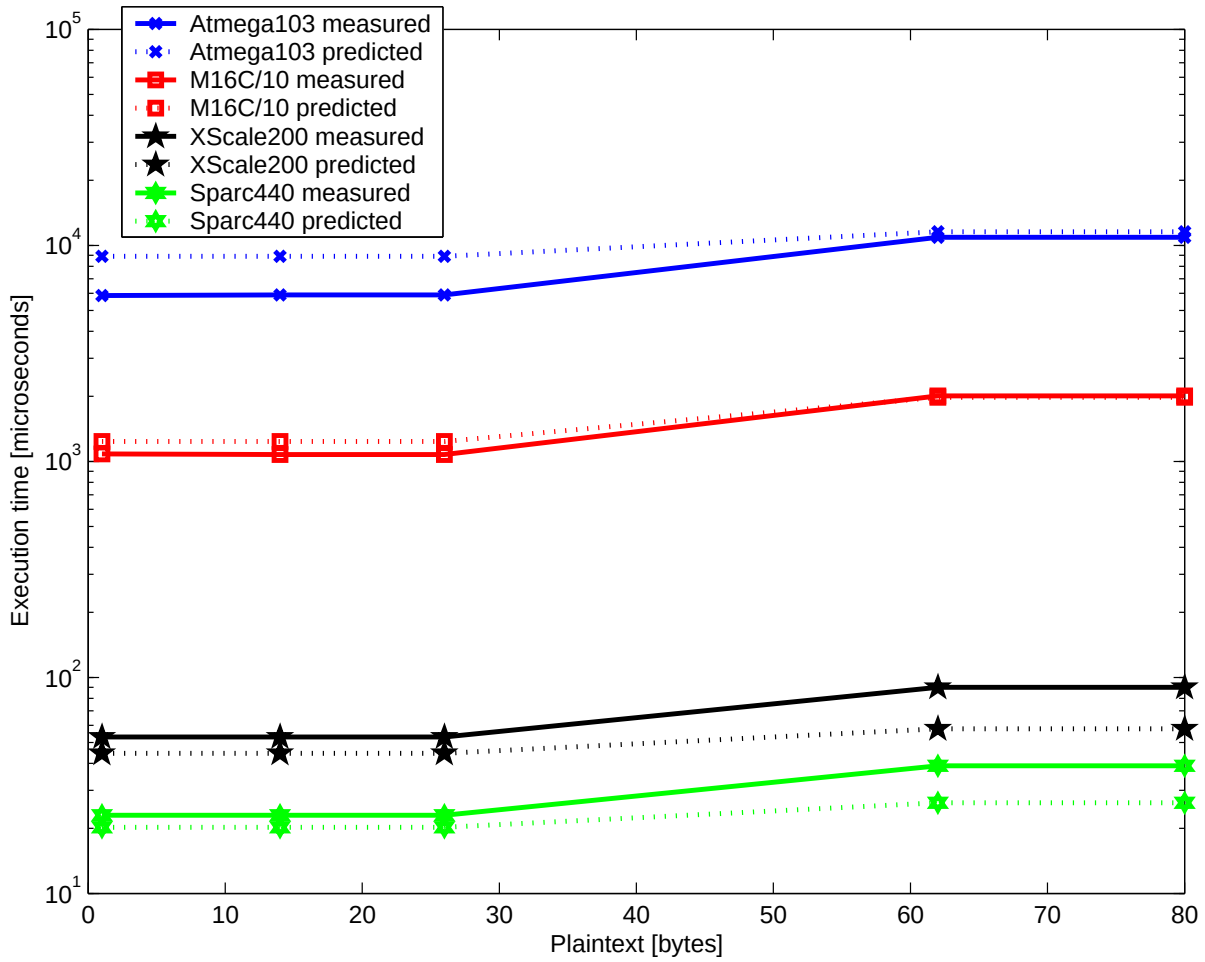

Figure 5: Measured and predicted execution times for MD5

Similarly, the influence of the CISC vs. RISC architectures can be separated by considering the M16C/10 (CISC) and the other microcontrollers, which all are RISC architectures.

For example, for the MD5 the parameters a_{RISC} and b_{RISC} are:

Table 6: Parameters a_{RISC} and b_{RISC} for the the IDEA encryption algorithm

	a_{RISC}	b_{RISC}
RISC	3207	1661
CISC	77175	-103593

Using the model presented, one can predict the performance of a particular algorithm on a specific architecture even before the architecture is implemented. In Figure 5, the measured times and the predicted times are plotted as a function of the length of the plaintext for MD5 for a few of the architectures considered in this paper.

5. Related Work

Prior work has shown that public key cryptographic algorithms can be a viable solution for constrained high-end wireless devices [6]. RSA key generation on smart cards [20] further shows that the generation of up to 1024 bit prime numbers is costly both in terms of time and energy for embedded systems (~20 sec on a 3.57 MHz Infineon SLE66CX160S). Even if keys were pre-generated, communication of lengthy public keys as well as their storage for each sensor node in range adds to these costs. Multiply operations in cryptographic schemes as a potential source of power consumption has been evaluated on low-end microcontrollers [17]. A secure architecture for constrained systems (like sensors) has been implemented in SPINS [12].

Other papers have analyzed the timing of encryption algorithms on higher end machines such as the performance analysis of MD5 [21] where timing requirements on various high-end architectures have been shown and in [16] where various symmetric key ciphers'

performance have been measured in cycles and analyzed. Change in various processing times with changes in MIPS capability of a processor has been modeled [14]. Some symmetric and asymmetric key algorithms have been evaluated on higher end microprocessors on the basis of power consumption [7]. Cryptographic overhead for performance critical systems [22] using a hash, secret key and public key examples for high-end and one embedded architecture (16MHz Motorola 68K). Also, general benchmarks for speed have been computed on a Celeron processor [8]. Our work attempts to bridge the gaps by assessing the performance of algorithms on different platforms and evaluates the overhead of each algorithm on different architectures. To our knowledge, there is currently no published work that focuses on evaluation of different cryptographic algorithms on embedded architectures, particularly for low-end systems, such as 8-bit and 16-bit architectures.

6. Future Work

Our proposed model helps to extrapolate the performance of a algorithm on different platforms. This could be enhanced to consider individual operations in each algorithm and provide a generic model where performance of any algorithm on any platform can be extrapolated. Many ad-hoc network security protocol schemes suggest the use of a variety of cryptographic algorithms. The model could be scaled to estimate the performance of these schemes. There are also some fast encryption algorithms, such as SEAL 3.0 [17], TEA [18] or TREYFER [19], that show very good performance in software implementations. These could be evaluated. However, more information on the strengths of these algorithms is necessary.

7. Conclusion

In this paper, we presented a survey investigating the computational requirements for a number of cryptographic algorithms and embedded architectures. The measurements obtained cover a wide class of commonly used encryption protocols and determine the impact of embedded architectures on performance. Our experiments indicate a mostly uniform cycle overhead for each word size (8/16/32 bit) but differences between the three word-size classes. The impact of caches is negligible while ISA support is limited to specific effects on certain algorithms. Specifically, we were surprised to find that RC4 outperforms RC5 on encryption in low-end processors, compared to the choice of RC5 for current sensor networks [12]. Hashing techniques require almost an order of a magnitude higher overhead.

We also derived a model to assess the computational overhead of embedded architectures for encryption protocols in general. Our analytical model assesses the impact of arbitrary embedded architectures as a multi-variant function for each encryption scheme depending on processor frequency, word width, ISA type and specific ISA support. This allows one to project computational limitations and determine the threshold of feasible encryption schemes under a set of the constraints for an embedded architecture.

Overall, our results are not only valuable to assess the feasibility of encryption schemes for arbitrary embedded architectures, but they also provide the basis for modeling encryption overheads across platforms.

8. References

[1] R. Rivest, "The RC5 encryption algorithm", in Proceedings of the 1994 Leuven Workshop on Fast Software Encryption, pages 86-96, Springer-Verlag, 1995, http://citeseer.nj.nec.com/rivest95rc.html

[2] B. Schneier, "Applied Cryptography", Second edition, John Wiley & Sons, 1996.

[3] R. . Rivest, "The MD5 Message-Digest Algorithm", IETF RFC 1321, April 1992. ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt

[4] D. Eastlake and P. Jones. "US Secure Hash Algorithm 1 (SHA1)",IETF RFC 3174, Sept. 2001. ftp://ftp.rfc-editor.org/in-notes/rfc3174.txt

[5] Sun microsystems, "UltraSPARCTM II Microprocessor", http://www.sun.com/processors/UltraSPARC-II/PBN-0140.pdf

[6] M. Brown, D. Cheung, D. Hankerson, J. Hernandez, M. Kirkup, A. Menezes, "PGP in Constrained Wireless Devices", in Proceedings of the 9th USENIX Security Symposium, Denver Colorado, pp. 247-261, Aug. 2000.

[7] D. Carman, P. Kruus, B. Matt, "Constraints and approaches for distributed sensor network security", NAI Labs technical report #00-010, Sept 2000, http://download.nai.com/products/media/nai/zip/nailabs-report-00-010-final.zip.

[8] W. Dai, "Crypto++ 4.0 Benchmarks", http://www.eskimo.com/~weidai/benchmarks.html

[9] C. Madson, "The Use of HMAC-SHA-1-96 within ESP and AH", IETF RFC 2404, Nov. 1998, http://www.ietf.org/rfc/rfc2404.txt

[10] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", IETF RFC 2401, Nov. 1998, ftp://ftp.rfc-editor.org/in-notes/rfc2401.txt

[11] The International PGP Home Page, http://www.pgpi.org

[12] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, D. Culler, "SPINS: Security Protocols for Sensor Networks", Proc. 7th Ann. Intl. Conf. Mobile Computing and Networking (MobiCom 2001), pp. 189-199, 2001.

[13] "LAN MAN Standards of the IEEE Computer Society. Wireless LAN medium access control (MAC) and

physical layer(PHY) specification IEEE Standaard 802.11, 1997 Edition," 1997.
[14] O. S. Elkeelany, M. M. Matalgah, K. P. Sheikh, M. Thaker, G. Chaudhry, D. Medhi, and J. Qaddour, "Performance Analysis of IPSec Protocol: Encryption and Authentication", IEEE Communications Conference (ICC 2002), pp. 1164-1168, 2002.
[15] C. Schnorr, Efficient signature generation by smart cards, Journal of Cryptology, vol. 4, pages 161-174, 1991.
[16] J. Burke, J. McDonald, T. Austin, "Architectural support for fast symmetric-key cryptography", ASPLOS-IX, 2000, pp. 178-189
[17] P. Rogaway, D. Coppersmith "A Software-Optimized Encryption Algorithm", Proceedings of the 1st International Workshop on Fast Software Encryption, Springer LNCS, Vol. 809, 1994, pp. 56-63.
[18] D. Wheeler, M. Needham, "TEA, a Tiny Encryption Algorithm", Fast Software Encryption: Second International Workshop", Springer LNCS, Vol. 1008, 1994, pp. 14-16.
[19] G. Yuval, "Reinventing the Travois: Encryption/MAC in 30 ROM Bytes" in Proc. 4th Workshop on Fast Software Encryption, Springer LNCS, Vol. 1267, 1997, pp. 205-209.
[20] C. Lu, A. Santos, F. Pimenetel, "Implementation of fast RSA key generation on smart cards", ACM Symposium on Applied Computing, 2000.
[21] J. Touch, "Performance analysis of MD5", Proceedings of the ACM SIGCOMM, Oct. 1995, pp. 77-86
[22] W. Freeman, E. Miller, "Experimental analysis of cryptographic overhead in performance-critical systems", 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1999, pp. 348-357.
IPAQ devices from Compaq, http://www.compaq.com/products/iPAQ/