CS 714 – Real Time Computing Systems

Final Report

Analysis of Timing Variability for Encryption Algorithms for the Atmega platform

Dinesh Dasarathan
ddasara@unity.ncsu.edu

Abstract

Security in Embedded systems is emerging as a growing issue in recent times. Limitations in processing power and communications bandwidth and the mismatch between wide arithmetic for security(32 bit word operations) and embedded data bus widths(8 or 16 bits) offer a great challenge in implementing encryption algorithms for embedded systems.

The variability of execution times is of particular interest to Real Time systems. Variations of execution times in small embedded systems may be due to the software mechanisms used by the compiler to implement certain features that are not present, such as caches that are prevalent in a normal processor. The results that come out of the variability analysis can be valuable in the analysis of worst-case execution times suited for schedulability analysis in the context of real-time systems.

Problem Statement

The analysis of how execution times for different cryptographic algorithms vary in the Atmega 128 processor forms the basis of this project. The results could be useful in the analysis of worst-case execution times that form an important discussion in determining the applicability of real-time systems.

Problem Details

The cryptographic algorithms used are IDEA, RC4 and RC5. 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. IDEA (International Data Encryption Algorithm) is a symmetric-key block cipher that operates on 64 bit plaintext blocks. The algorithm primarily includes operations from three algebraic groups: XOR, addition modulo 216, multiplication modulo 216+1. RC5 is a fast symmetric block cipher with a variety of parameters: block size, key size and number of rounds. It uses the XOR, addition and rotation operations.

The Atmega128 is at the high end of the AVR family's performance spectrum. It is a RISC architecture featuring 8 bit native word size, 32 general purpose registers and support for 16 bit operations. It features a 2 stage pipeline and has a 2-cycle multiply instruction.

Progress

All 5 encryption algorithms (IDEA, RC4, RC5, SHA-1 and MD5) have been compiled for the Atmega-128 platform and executed on it. The *avr-gcc* tool was used to compile the programs into .hex format and the *uisp* tool was used to deploy them onto the processor. UART functions to read and write data were written and used. But I had a problem with transferring strings through the UART, and so I had to manage with only character transfers. All integers and longs are displayed in hexadecimal format.

These were the results I obtained. The results are in the number of clock cycles taken to execute one run of the algorithm.

Results

S.No.	Algorithm	Size	Action	Original results	My results
1	MD5	1-26	Digest	22616	20325
		62-80	Digest	43456	38329
2	SHA-1	1	Digest	63303	54738
		3	Digest	63199	54867
		56	Digest	60276	49244
		64	Digest	126224	110350
3	RC5	16	Init	38564	28244
			En	6772	5886
			De	6736	5842
4	RC4		Init	54392	3875
			En	6300	919
5	IDEA	16	Init En	6411	4572
			Init De	35717	29277
			En	6000	5270
			De	6000	5273

The original results refer to results obtained by running the algorithms on a Windows based system using WinAVR for compiling and deploying the algorithms on the Atmega platform. I have programmed the Atmega128 with a 3.69 MHz clock and the serial port's baud rate is set to 9200 bps. The results are displayed in minicom, a hyperterminal for Linux.

The results that I have obtained show that the algorithms perform better than their original versions that were compiled using WinAVR. This is because I have compiled them using the gcc option –O3 that generates highly optimized code. We can see that the RC4 has more than a magnitude of difference, but yields the correct result. When no optimization is used, the RC4 yields results similar to that of the original values.

Variability

Variability in IDEA:

I ran the IDEA algorithm with a wide range of inputs (the input being the key for initiating the encryption), and achieved variations. The variation is due to the series of multiplications done by the algorithm.

For one multiplication, if either operand is zero, the variability is quite high, and with all 34 of the multiply operations have either of their operands to be zero, then the variability is as high as 40%. If it is guaranteed that there are no zero operands, then the variability is 2.58%. This is due to the change in control flow inside the multiplication routine. The routine is shown below for convenience:

```
/* multiplication using the Low-High algorithm */
uint32_t mul(uint32_t a,uint32_t b)
{
        long p;
        uint32_t q;
                if(a==0)
                        p=maxim-b;
                else
                        if(b==0)
                                p=maxim-a;
                        else {
                                q=(uint32_t)a*(uint32_t)b;
                                p=(q & one) - (q>>16);
                                if(p<=0)
                                        p=p+maxim;
                        }
        return (uint32_t)(p&one);
}
```

If a or b is zero, then multiplication does not need to be done. This amounts to the high variability found therein. Otherwise, the variability is due to p less than or equal to zero, which is not so high. The multiplication operation as such does not induce any variability in the Atmega128, since it takes a constant 2 cycles all the time, whatever the input is.

Variability of RC4 and RC5:

The RC4 and RC5 algorithms are completely deterministic, i.e., they show no variability, whatever be the input. The RC4 was found deterministic even in the M16C, but the RC5 showed a variability of 2.5% in the M16C processor. This is because the M16C has a multiple shifter, that takes (1+m) cycles for each multiple shift, where m depends on the length of the shift. This induces the variability found in RC5.

But, the Atmega128 does only a single shift per clock. The RC5 uses shift operations that amount to a constant number of shifts. The corresponding code is shown below:

```
#define w 32
#define ROTL(x,y) (((x)<<(y&(w-1)))|((x)>>(w-(y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1)))|((x)<<(w-(y&(w-1)))))
```

where x and y depend on the input.

For ROTL, the number of left shifts is variable, depending on the input, and so is the number of right shifts. This induces the variability found in M16C. But the total number of shifts per ROT* operation is always 32, and so it takes constant time in the Atmega128 to execute these. Hence, the RC5 is completely deterministic for the Atmega128 processor.

If the code had been like this, we could find variability in the Atmega128, since the number of shifts is variable.

```
#define ROTL(x,y) (((x)<<(y&(w-1)))|((x)>>((y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1)))|((x)<<((y&(w-1)))))
```

Variability of MD5:

The variability of the MD5 is dependant only on the input size and not on the type of input. For every 56+x where x is multiple of 64, there is a constant increase in clock ticks by 18004. This is because of an extra iteration of a for loop in the code. In each such band, the max variability can be 598 clocks for the first 8 iterations.
Hence, the variability can be found as,

0-119: 0%

120-183: 1.07%

184-247: 0.81%

And as input size increases further, variability keeps on decreasing.

Variability of SHA1:

Here too, input size alone determines the change in execution times. The change in input for a given input size does not determine the execution time. For every 64 iterations of input size, the clock ticks increases by 55670 ticks. Within each band, the variation is 74 ticks per increase in i/p size.

64-127 : 0.126%

128-192: 0.064% , and variability keeps on decreasing.

Final Results

Algorithm	*Maximum Variability*	*Comments*
IDEA	**40%**	IDEA depends on multiply operations. Variability for the special case (either operand is zero) is 40%. But this is rare, and the general variability excluding this case is 2.5%
RC4	**0%**	Completely deterministic
RC5	**0%**	RC5 depends on shift operations. Also completely deterministic, since constant number of shifts take constant number of cycles to execute.
MD5	**1.07%**	Depends only on Input size, and not on type of input.
SHA-1	**0.12%**	Also depends on Input size – meager variability found.

References

[1] "Encryption Overhead for Sensor Networks and Embedded Systems: Modeling and Analysis" by Ramnath Venugopalan, Prasanth Ganesan, Pushkin Peddabachagari, Alexander Dean, Frank Mueller and Mihail Sichitiu in Conference on Compiler, Architecture and Synthesis on Embedded Systems (CASES'03), Oct/Nov 2003, pages 188-197

[2] "Analyzing and Modeling Encryption Overhead for Sensor Network Nodes" by Prasanth Ganesan, Ramnath Venugopalan, Pushkin Peddabachagari, Alexander Dean, Frank Mueller and Mihail Sichitiu in Workshop on Wireless Sensor Networks and Applications (WSNA '03) with MobiCom'03, Sep 2003, pages (accepted).

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

[4] J.Burke, J.McDonald, and T.Austin, "Architectural support for fast symmetric-key cryptography," in Proc. of ASPLOS-IX, 2000, pp. 178-189.