CS 714 – Real Time Computing Systems

Project proposal

Analysis of Timing Variability for Encryption Algorithms for the Atmega platform

Dinesh Dasarathan
ddasara@unity.ncsu.edu

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.

Progress made till now

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	53948
		3	Digest	63199	54236
		56	Digest	60276	54092
		64	Digest	126224	54092
3	RC5	16	Init	38564	28244
			En	6772	5886
			De	6736	5842
4	RC4		Init	54392	3879
			En	6300	816
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. But still, my RC4 and SHA-1 versions do not generate results close to the original results.

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. I had variations of 1.33 % for encryption and 2.7% for decryption, but these results are not yet validated, since they were obtained by only testing a wide range of inputs, and there is no substantial evidence that no more variation can be found.

Pitfalls

1) String transfers: I could not transfer strings through the UART. I am instead transferring single characters and getting the desired o/p.
2) RC4: My RC4 results do not match with the original results. They, in fact, have an order of 10 difference from the original results.
3) The SHA-1 algorithm does not perform as expected for the 64 bit case.

Remainder of the Project

The remainder of the project involves finding variability for the 5 algorithms, check its validity, and correct the above mentioned problems. Finding variability in execution times can be found by analyzing the Control flow of the algorithms. This involves providing different input variations that can traverse through the different control flows of the algorithms and find variations in execution times among them. Finding variability using a wide range of inputs does not validate the results, and therefore, control flow analysis (eg: traversing the "if" part once, and traversing the "then" part next) has to be employed to validate the results.

Weekly deadlines

Week 1: Finding and validating the variability among IDEA and RC5.
Detecting why string transfers do not work, and correcting them.
Week 2: Finding and validating the variability of MD5.
Correcting the RC4 and SHA-1 to give results close to the original results.
Week 3: Finding and validating the variability of RC4 and SHA-1.
If the problem related to string transfers is solved by this time, I shall look into providing a generic encryption algorithm that for any input string runs it on all algorithms and finds out the best encryption algorithm for that particular string.