Evaluation of Cryptology Algorithms in IoT

Project Report

Contributers

Münteha Nur Bedir
2035731
munteha.bedir@ceng.metu.edu.tr

Ülkü Meteriz
2036093
ulku.meteriz@ceng.metu.edu.tr

CONTEXT

1. Problem Statement

The Internet of Things, shortly IoT, is a form of network consisting of embedded devices, sensors, actuators or even softwares on small boards, such as Arduino and Rasperry Pi, which can exchange data with each other. Next generation of technology seems to be depend on IoT devices. The usage of devices that capable of connecting network increased 31% from 2016 to 8.4 billion in 2017 [1]. Additionally, specialist in IoT area foresees that the IoT devices will be used in about 30 billion objects by 2020 [2]. This means that, IoT devices will know everything about users's daily life which mosr probably contains sensitive information of users. In order not to leak sensitive information of its users, security is an important aspect for IoT devices.

Current approaches to security mainly depends on encryption of the data and the secrecy of the keys that are used in encryption. Therefore, IoT devices should be capable of encrypt and decrypt data to maintain security. Due to lightweight infrastructure of IoT devices, robost and secure encryption algorithms and secure hash functions may not be optimal for these lightweight design. In this technical report, the main purpose is providing an insight and a guideline to interested researchers and IoT developers about performance of widely used expensive and lightweight encryption algorithms and secure hash functions on commonly used IoT device, Arduino Uno.

2. Related Work

There are excessive number of studies on IoT devices and their security aspects. Many researchers try to find new encryption algorithms and proper protocols to be used in communication among IoT devices and they evaluate their performance results. One and not the least drawback of proposing new encryption algorithms is that these algorithms are not intensively tested by cryptanalysts, so their reliability and robostness cannot be trusted. Suprisingly, there are limited number of studies that measures the performance of commonly used reliable encryption algorithms. One of them is made by Hannes Tschofenig who is a developer in ARM microprocessors. In his papers, he measures the performance of SHA, AES and many other commonly used algorithms on ARM microprocessors.[3]

In the paper named *"Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices"*[4], Balasch *et al.* measured the performance of several the hash functions in ATtiny devices, which contains ATtiny 8-bit AVR microcontroller. [5]

In Yalçın T. and Kavun E.B's study[6], they implement and measured the performance of the sponge-based authenticated encryption algorithm, SpongeWrap , on ubiquitous devices.

In the paper of Darlis, D. *et al.* [7] , they implement BLOWFISH encryption algortihm which is a lightweight encryption algorithm on FPGA, which is also a widely used embedded

device, using VHDL language. In their paper, all implementation steps including key expansion and data encryption are clearly explained. By reducing the round count of Fiestel structure, they realized that the total encryption time is also reduced, while avalanche effect is not affected significantly.

3. Implementation

3.1.Used Hardware

Figure 1: *Arduino UNO R3*

We prefered Arduino Uno to measure performance of various Crypto algorithms. Arduino boards are one of the most popular boards in IoT world and the uno is the most commonly used board within whole Arduino family. [8]

Memory : The microprocessor has two main memory types as data and program memory.

Program (Flash) Memory : Flash memory is a reprogrammable memory for program storage.

SRAM Data Memory : SRAM Data Memory is used by Register File, I/O operations and global and local variables.

EEPROM Data Memory : It is organized as a separate data space, in which single bytes can be read and written.[9]

Power : The Arduino Uno board can be powered via the USB connection or with an external power supply(can be AC or DC). In case of one of them are connected, the power source is selected automatically.[8] In normal conditions, the microprocessor uses 0.2mA in active mode, 0.1µA in power-down mode. [9]

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz

Table1: *Technical Specs of Arduino Uno* [8]

3.2. Used Integrated Development Environment

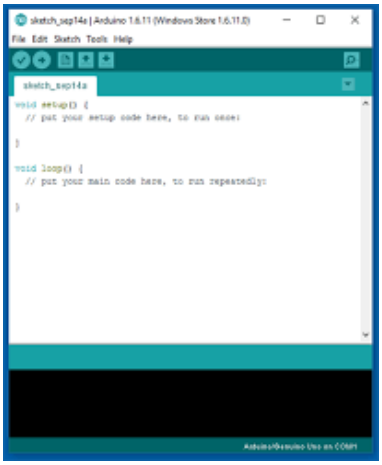

Figure2: *Arduino Development Software*

The Arduino Uno can be programmed in Arduino development Software. There are online and desktop versions of the IDE. The Arduino uses a language similar to C/C++

programming language. It is like simplified version of C/C++. Programs can be uploaded to the board by using the IDE.

3.3. Used Libraries

To measure time, “Arduino cryptograpy library” is used for optimized cryptographic algorithms . The library is distributed under the terms of the MIT license. It also contains examle Arduino codes showing the usage.[10]

To measure memory usage, “MemoryFree” library is used. The library is distributed under the terms of "THE BEER-WARE" license.[11]

3.4. Used Algorithms

3.4.1. Block Ciphers

3.4.1.a. AES:

AES standardized by NIST in 2001, is a slightly changed version of the Rijndael cipher designed by cryptographers Joan Daemen and Vincent Rijmen.

AES is a symmetric block cipher with a block length of 128 bits. It allows for three different key lengths: 128, 192, or 256 bits. AES uses a substitution permutation network instead of Fiestel network. Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. [12]

3.4.1.b. ASCON:

Ascon is a family of authenticated encryption algorithms and the finalist of the CAESAR competition in 2016, designed by cryptographers Christoph Dobraunig, Maria Eichlseder, Florian Mendel and Martin Schläffer. The Ascon family was designed to be lightweight and easy to implement, even with additional countermeasures against side-channel attacks.[13]

Ascon has been designed to do a minimum number of operations while still maximizing the concurrency of these operations. Therefore, Ascon is best used in the case of size and security are significant, but high performance is also required. Ascon is applicable where many devices with limited resource communicate with a server or each other. Ascon is designed based on the sponge methodology. The permutation of Ascon uses substitution permutation network (SPN), which supplies good diffusion with low cost.[14]

3.4.2. Secure Hash Functions

3.4.2.a. SHA:

The SHA-256 is a cryptographic hash function operates on a 512-bit message block and a 256- bit intermediate hash value. It is essentially a 256-bit block cipher algorithm which encrypts the intermediate hash value using the message block as key. A message is processed by blocks of 512 = 16 × 32 bits, each block requiring 64 rounds.

The SHA-512 is a cryptographic hash function operates on a 1024-bit message block and a 512-bit intermediate hash value. It is essentially a 512-bit block cipher algorithm which encrypts the intermediate hash value using the message block as key. SHA-512 is a variant of SHA-256 which operates on eight 64-bit words.[15]

3.4.2.b. BLAKE2 :

BLAKE2 is the cryptographic hash function improved version of the SHA-3 finalist BLAKE optimized for speed in software.

BLAKE2 has two types: *BLAKE2B* is optimized for 64-bit platforms and produces digests of any size between 1 and 64 bytes. *BLAKE2S* is optimized for 8- to 32-bit platforms and produces digests of any size between 1 and 32 bytes. On 64-bit platforms, BLAKE2 is often faster than MD5, yet provides security similar to that of SHA-3.[16]

3.4.3. Public Key Algorithms

3.4.3.a. Diffie-Hellman key exchange:

The Diffie Hellman key exchange is a breakthrough in public-key cryptography of the 1970's, designed by Whitfield Diffie and Martin Hellman in their leading-edge paper "New Directions in Cryptography".

The security of the Diffie Hellman key exchange based on the complexity of computing discrete logarithms only knowing public values. However, both receiver-side and sender-side are able to derive a shared secret key from one side's public key and the other's private key.[17]

3.4.3.b. Curve25519:

Curve25519 designed by Daniel J. Bernstein is a most improved version of elliptic-curve Diffie-Hellman function applicable for various cryptographic applications. Each Curve25519 user has a 32- byte secret key and a 32-byte public key. A hash of the shared secret Curve25519 is used as the key for a secret-key authentication system. [18]

4.Evaluation

For evaluating the implementation of mentioned algorithms, we measure the memory usage, the execution time and the energy consumption for each encrypted byte.

Memory usage includes the flash memory, stack memory and dynamic memory which is the heap region. The maximum memory size for flash memory is 32256 bytes which is 31.5 KB. The maximum memory size for the dynamic memory is 2048 bytes, which is 2 KB.

To calculate the energy consumption , we used the following formulas:

Operation current of microprocessor in active mode: 0.2mA
Voltage supplied by USB port: 5V
Unit power consumption: 0.2mA * 5V = 1 mW
Energy Consumption: 1mW * elapsed time

4.1. Block Ciphers

4.1.a. AES :

Key size: 128 bits

Memory:

Flash memory usage: Program code uses 8270 bytes (25%) of program storage space.
Heap memory usage: Global variables use 886 bytes (43%) of dynamic memory, leaving 1162 bytes for local variables.
Stack memory usage: 1156 bytes out of 1162 bytes.

Time:

AES-128-ECB Key Setting: 250.06µs per operation, 3999.03 per second
AES-128-ECB Encryption: 33.56µs per byte, 29796.89 bytes per second
AES-128-ECB Decryption: 67.50µs per byte, 14814.81 bytes per second

Key size: 192 bits

Memory:

Flash memory usage: Program code uses 8270 bytes (25%) of program storage space.

Heap memory usage: Global variables use 918 bytes (44%) of dynamic memory, leaving 1130 bytes for local variables.
Stack memory usage: 1124 bytes out of 1130 bytes.

Time:

AES-192-ECB Key Setting: 277.22µs per operation, 3607.19 per second
AES-192-ECB Encryption: 40.31µs per byte, 24806.36 bytes per second
AES-192-ECB Decryption: 81.77µs per byte, 12228.85 bytes per second

Key size: 256 bits

Memory:

Flash memory usage: Program code uses 8566 bytes (26%) of program storage space.
Heap memory usage: Global variables use 950 bytes (46%) of dynamic memory, leaving 1098 bytes for local variables.
Stack memory usage: 1092 bytes out of 1098 bytes.

Time:

AES-256-ECB Set Key Setting: 434.55µs per operation, 2301.23 per second
AES-256-ECB Encryption: 47.06µs per byte, 21247.64 bytes per second
AES-256-ECB Decryption: 96.05µs per byte, 10411.50 bytes per second

4.1.b. ASCON:

Key : 128 bits

Memory:

Flash memory usage: Program code uses 12152 bytes (37%) of program storage space.
Heap memory usage: Global variables use 1078 bytes (52%) of dynamic memory, leaving 970 bytes for local variables.
Stack memory usage: 964 bytes out of 970 bytes.

Time:

Key Setting: 738.77µs per operation, 1353.60 per second
Encryption: 42.71µs per byte, 23413.89 bytes per second
Decryption: 43.07µs per byte, 23216.81 bytes per second
Adding Authentication Data: 41.81µs per byte, 23919.69 bytes per second
Computing Tag: 729.03µs per operation, 1371.69 per second

Function Name	Flash Memory (byte)	Heap Memory (byte)	Stack Memory (byte)	Key Setting Time (per operation)	Encryption Time (per byte)	Decryption Time (per byte)	Energy Consumption (per byte)
AES128	8270	886	1156	250.06µs	33.56µs	67.50µs	101.06nJ
AES192	8270	918	1124	277.22µs	40.31µs	81.77µs	122.08nJ
AES256	8566	950	1092	434.55µs	47.06µs	96.05µs	143.11nJ
ASCON	12152	1078	964	738.77µs	42.71µs	43.07µs	85.78nJ

Table2: *Performance Measurements of Block Ciphers*

As shown in the Table2, when key size is increased, memory usage, operation time and energy consumption also increases for AES algorithm. Cost of providing more security by using AES is an expensive option in the sense of resource usage . Since ASCON is designed as lightweight algorithm, its energy consumption and operation time are less than AES.

4.2. Secure Hash Function

4.2.a. SHA:

SHA256

Memory:

Flash memory usage: Program code uses 9552 bytes (29%) of program storage space.
Heap memory usage: Global variables use 802 bytes (39%) of dynamic memory, leaving 1246 bytes for local variables.
Stack memory usage: 1240 bytes out of 1246 bytes.

Time:

Hashing: 43.93µs per byte, 22763.00 bytes per second

SHA512

Memory:

Flash memory usage: Program code uses 15176 bytes (47%) of program storage space.

Heap memory usage: Global variables use 934 bytes (45%) of dynamic memory, leaving 1114 bytes for local variables.
Stack memory usage: 1108 bytes out of 1114 bytes.

Time :
Hashing: 124.72µs per byte, 8018.21 bytes per second

4.2.b. BLAKE2:

BLAKE2B:

Memory:

Flash memory usage: Program code uses 24136 bytes (74%) of program storage space.
Heap memory usage: Global variables use 866 bytes (42%) of dynamic memory, leaving 1182 bytes for local variables.
Stack memory usage: 1110 bytes out of 1182 bytes.

Time:
Hashing: 65.34µs per byte, 15304.46 bytes per second.

BLAKE2S:

Memory:

Flash memory usage: Program code uses 24136 bytes (74%) of program storage space.
Heap memory usage: Global variables use 866 bytes (42%) of dynamic memory, leaving 1182 bytes for local variables.
Stack memory usage: 1110 bytes out of 1182.

Time:
Hashing: 65.3µs per byte, 15304.46 bytes per second

Function Name	Hashing Time (per byte)	Flash Memory Usage (bytes)	Heap Memory Usage (bytes)	Stack Memory Usage (bytes)	Energy Consumption (per byte)
SHA256	43.93µs	9552	802	1240	43.93 nJ
SHA512	124.72µs	15176	934	1108	124.72 nJ
BLAKE2B	65.34µs	24136	866	1100	65.34 nJ
BLAKE2S	65.3µs	24136	866	1110	65.3 nJ

Table3: *Performance Measurements of Secure Hash Functions*

As can be seen in Table3 above, both time and memory measurements of the SHA secure hash functions vary depending on the output size. As expected, outputting 512-bit hash value is more expensive than outputting 256-bit hash value.

Since BLAKE2B and BLAKE2S algorithms are very similar to each other as mentioned in the ***3.4 Used Algorithms*** section, their memory, time and energy consumptions are almost the same.

BLAKE2 secure hash functions might be considered as an option between SHA512 and SHA256 in the manner of source usage.

4.3. Public-Key Algorithms

4.3.a. Diffie-Hellman Key Exchange:

Memory:

Flash memory usage: Program code uses 2746 bytes (8%) of program storage space.
Heap memory usage: Global variables use 242 bytes (11%) of dynamic memory, leaving 1806 bytes for local variables.
Stack memory usage: 1800 bytes.

Time:
Public-key generation: 16000µs
Shared secret key generation: 300000µs

4.3.b. Curve25519:

Memory:

Flash memory usage: Program code uses 11298 bytes (35%) of program storage space.
Heap memory usage: Global variables use 1131 bytes (55%) of dynamic memory, leaving 917 bytes for local variables.
Stack memory usage: 881 bytes out of 917 bytes.

Time:

Public-key generation: 3240540 µs
Shared secret-key generation: 3240548 µs

Name	Flash Memory Usage	Heap Memory Usage	Stack Memory Usage	Public-Key Generation Time	Secret-Key Generation Time	Energy Consumption
Diffie-Hellman	2746	242	1800	16000µs	300000µs	316000mJ
Curve 25519	11298	1131	881	3240540µs	3240548µs	6.481088 mJ

Table4: *Performance Measurements of Public-Key Algorithms*

As shown in the Table4, Diffie-Hellman algorithm uses less resource than Curve25519, since Curve2519 algorithm is based on elliptic curves, and it requires more complex computations.

5. Conclusion

In this report, we measured performance of different types of cryptographic algorithms that can be classified as three categories: Block Ciphers, Secure Hash Functions and Public-Key Algorithms. While measuring performance, we took into consideration memory usage, time and energy consumption. After testing performance and taking the findings, we explained the reasoning behind the differences. As a result of this study, we formed a guideline for the ones that are interested in developing a secure IoT systems by using Arduino Uno.

6. References

1. Köhn, R. (2018, February 16). Online-Kriminalität: Konzerne verbünden sich gegen Hacker. Retrieved from http://www.faz.net/aktuell/wirtschaft/diginomics/grosse-internationale-allianz-gegen-cyber-attacken-15451953-p2.html?printPagedArticle=true#pageIndex_1

2. Nordrum, Amy (18 August 2016). "Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated". *IEEE*.

3. Tschofenig, H., & Pegourie-Gonnard, M. (2015). Implementations & Performance , Performance of State-of-the-Art Cryptography on ARM-based Microprocessors. *NIST Lightweight Cryptography Workshop Session VII*.

4. Balasch, J., Ege, B., Eisenbarth, T., Gérard, B., Gong, Z., Güneysu, T., . . . Maurich, I. V. (2013). Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices. *Smart Card Research and Advanced Applications Lecture Notes in Computer Science,* 158-172. doi:10.1007/978-3-642-37288-9_11

5. Tiny817 Product Family. (n.d.). Retrieved from http://www.microchip.com/promo/tiny817

6. Yalçın, T., & Kavun, E. B. (2013). On the Implementation Aspects of Sponge-Based Authenticated Encryption for Pervasive Devices. *Smart Card Research and Advanced Applications Lecture Notes in Computer Science,* 141-157. doi:10.1007/978-3-642-37288-9_10

7. Prasetyo, K. N., Purwanto, Y., & Darlis, D. (2014). An implementation of data encryption for Internet of Things using blowfish algorithm on FPGA. *2014 2nd International Conference on Information and Communication Technology (ICoICT).* doi:10.1109/icoict.2014.6914043

8. Arduino Uno. (n.d.). Retrieved from https://store.arduino.cc/usa/arduino-uno-rev3

9. ATmega328/P. (n.d.). Retrieved from http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

10. Arduino Crytography Library. (2018, April 27). Retrieved from https://rweather.github.io/arduinolibs/crypto.html

11. MemoryFree. (2012, November 4). Retrieved from https://github.com/sudar/MemoryFree

12. Advanced Encryption Standard. (2001, November 26). Retrieved from https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

13. ASCON. (n.d.). Retrieved from https://ascon.iaik.tugraz.at/

14. Dobraunig, C., Eichlseder, M., Mendel, F. & Schlaffer, M. (2016, September 15). Ascon v1.2 Submission to the CAESAR Competition. Retrieved from http://competitions.cr.yp.to/round3/asconv12.pdf

15. Descriptions of SHA-256, SHA-384, and SHA-512. (n.d.). Retrieved from http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf

16. Aumasson, J. P., Neves, S.,O'Hearn, Z. W. &Winnerlein, C. (2013, January 29). BLAKE2: simpler, smaller, fast as MD5. Retrieved from https://blake2.net/blake2.pdf

17. Scarvalone, M. (2009, July 17). RSA Encryption and Diffie Hellman Key Exchange. Retrieved from https://www.math.uchicago.edu/~may/VIGRE/VIGRE2009/REUPapers/Scarvalone.pdf

18. Bernstein, D. J. (2006, February 9). Curve25519: new Diffie-Hellman speed records. Retrieved from https://cr.yp.to/ecdh/curve25519-20060209.pdf