16+

International Research Conference on Technology, Science, Engineering & Economy

SCIENTIFIC PUBLIC ORGANIZATION «PROFESSIONAL SCIENCE»

Seattle, USA, 2020

UDC 330-399
LBC 60

Editors
Natalya Krasnova | Managing director SPO “Professional science”
Yulia Kanaeva | Logistics Project Officer SPO “Professional science”

International Research Conference on Technology, Science, Engineering & Economy: Conference Proceedings, February 28th, 2020, Seattle, USA. SPO “Professional science”, Lulu Inc., 2020, 78 p.

ISBN 978-1-67819-457-4

Presenters outline their work under the following main themes: education, equality and development, pedagogy, language and culture in education, principles of environmental health, physiology, economics, finance & accounting.

The conference is well attended by representatives from more than 5 universities with participation of higher education institutional policymakers, governmental bodies involved in innovating, deans and directors, educational innovators, university staff and umbrella organizations in higher education.

www.scipro.ru

UDC 330-399
LBC 60

TABLE OF CONTENTS

SECTION 1. EDUCATION, EQUALITY AND DEVELOPMENT 4

KOZHATAYEVA B.B., MATENOVA G.B., ALIBEKOVA A.O. WAYS OF DEVELOPMENT ENGLISH TEACHER'S COMMUNICATIVE COMPETENCE 4

MUSANTAEVA M.A., UMIRZAKOVA M.A., RISKELDIEVA J.A., USIPBAEVA G.T., AULBEKOVA ZH.S. DIGITAL TECHNOLOGY: "THE METHOD OF VISUAL APPLICATION OF MEANS OF ELECTRONIC TEXTBOOKS AND VIDEO SELECTION IN TEACHING RUSSIAN LANGUAGE AND LITERATURE 10

SECTION 2. PEDAGOGY, LANGUAGE AND CULTURE IN EDUCATION 17

ZAIKA L.A. MODERN COMPETENCES OF THE TEACHER OF THE 21ST CENTURY 17

SECTION 3. PSYCHOLOGY AND EDUCATION 24

ALEKSEEVA A.A., IONOVA I.S., OSTROVSKY YU.K., CHUMAKOVA M.KH. COLORING AS A TOOL TO STUDY THE PSYCHOLOGICAL CHARACTERISTICS OF A PERSON 24

SECTION 4. ECONOMICS, FINANCE & ACCOUNTING 28

CHERNIKOVA O.A., BARKOVA I.V., KURBANOVA M.V., CHIVER L.P. THE EFFECTIVENESS OF THE FUNCTIONING OF ORGANIZATIONS IN THE STAVROPOL TERRITORY 28

SECTION 5. SCIENCE, TECHNOLOGY, ENGINEERING AND MATHS 35

OZHEGIN E.V. MOTIVATION AS A KEY POINT FOR IMPROVING WORK SAFETY 35

SECTION 6. MEDICINE, LIFE-SCIENCE, BIOMEDICINES 40

KOZHEVNIKOVA U.V., PRONKIN N.N. 3D ORGAN PRINTING (3D BIOPRINTING) 40

SECTION 7. MECHATRONICS AND AUTOMATION OPERATIONS RESEARCH 44

LIU Z. SIMULATION OF THE PROCESS OF REDUCTION FIRING NICKEL OXIDE 44

SECTION 8. COMMUNICATION NETWORKS AND SECURITY 52

TEMIRBEKOVA ZH.E., PYRKOVA A.Y., TEMIRBEK ZH.E. USING MICROCONTROLLERS TO ENSURE DATA SECURITY 52

SECTION 9. FINANCIAL AND MANAGEMENT ACCOUNTING 61

YESIKOVA R.S. THE COMPETENCY MODEL AS A MODERN ORGANIZATION PERSONNEL MANAGEMENT TOOL 61

ZERNOVA L.E. SMALL AND MEDIUM-SIZED BUSINESSES: CONCEPT, ROLE AND CURRENT STATE IN THE RUSSIAN FEDERATION 70

SECTION 8. COMMUNICATION NETWORKS AND SECURITY

UDC 004

Temirbekova Zh.E., Pyrkova A.Y., Temirbek Zh.E. Using microcontrollers to ensure data security

Temirbekova Zhanerke Erlanovna
Senior lecturer, Faculty of Information Technology, Al-Farabi Kazakh national university, Almaty, Kazakhstan
Pyrkova Anna Yurevna
Professor, Faculty of Information Technology, Al-Farabi Kazakh national university, Almaty, Kazakhstan
Temirbek Zhansaya Erlanovna
Teacher, 120 gymnasium named after Mazhit Begalin, Almaty, Kazakhstan·

Abstract. *Nowadays, the number of Internet of Things (IoT) devices with a wide range of tasks is growing rapidly around the world. Many types of these devices collect data and, depending on their purpose, some data may be very sensitive to the user. This requires security on the device, which must protect the collected data and send it to the server via the Internet. This article describes some of the programming and debugging techniques developed using the Mbed platform running on the BLE Nano kit microcontroller. C ++ was used as the primary language instead of C to improve programming efficiency. This turned out to be a good choice, but it led to an increase in the size of the program. Thus, much of the optimization was spent on reducing the size of the program and data. The main purpose of this article is to describe the Advanced Encryption Standard (AES) symmetric encryption algorithm, its security and complexity, using the ARM Mbed special encryption library. In addition, the article demonstrated homomorphic encryption in C ++, it is a form of encryption that allows you to perform certain types of computations on encrypted text and generate an encrypted result that, when decoded, corresponds to the result of operations performed on simple texts.*

Keywords: *Internet of Things (IOT), BLE Nano kit microcontroller, Mbed platform, AES cryptography, Homomorphic encryption.*

In the recent decade, microcontroller is getting popular application in portable devices, embedded system and mobile platform due to its integrated architecture, rich functionalities and increasing processing power. The Bluetooth Low Energy (BLE) microcontroller used is RedBearLab's BLE Nano. According to their website, the BLE Nano kit is the smallest Bluetooth 4.1 Low Energy (BLE) development board in the market [1-4]. The core is Nordic nRF51822 (an ARM Cortex-M0 SoC plus BLE capability) [5] running at 16MHz with ultra-low power consumption.

Figure 1. BLE Nano kit microcontroller

Features:

- Very small BLE development board, only 18.5mm x 21.0mm
- Nordic nRF51822 ARM Cortex-M0 SoC supports both BLE Central and BLE Peripheral roles
- 2.4 GHz transceiver
- Ultra low power consumption
- Support voltage from 1.8V to 3.3V
- Software development using Mbed.org, GCC, Keil or Arduino
- Lots of libraries and examples available
- Easy firmware deployment with the MK20 USB board
- Work with our free Android App and iOS App

The BLE Nano kit was chosen not only for its low power consumption but also for its ease of use. The board comes with the MK20 USB board and several software development options, including an online option that was used for this work. To upload code, the BLE Nano is attached to the MK20 USB board and plugged in to a USB port where upon a drag-and-drop interface facilitates uploading code to the board over the USB connection. The BLE Nano can then be removed from the MK20 USB board and tested in a breadboard circuit. Many existing libraries and examples as well as available Android and IOS applications made this microcontroller a good choice to start with.

To prepare the BLE Nano, header pins were soldered onto the Nano and MK20 USB board. Mbed was the online compiler and IDE used in this work, so code from the mbed RBL Nano page was used to run a blink test on the BLE Nano and boatload the MK20.

Programming with mbed

Mbed is an online compiler and API (Application Program Interface) that was used to code this work. BLE Nano kit microcontroller is supported by the Mbed's hardware platform. Applications for the Mbed platform can be developed using the Mbed online IDE, a free online code editor and compiler. Only a web browser needs to be installed on the local PC, since a project is compiled on the cloud, i.e. on a remote server, using the ARMCC C/C++ compiler [6].

Mbed OS provides the Mbed C/C++ software platform and tools for creating microcontroller firmware that runs on IoT devices. It consists of the core libraries that provide the microcontroller peripheral drivers, networking, build tools and test and debug scripts.

The many available platform libraries are easy to use and well designed, especially for starters, resembling (vaguely) the simplicity you may have enjoyed when using Arduino libraries. Also, Mbed encourages and eases sharing with others your own libraries, or any piece/snippet of code. You have to register a username in the platform from the very beginning that will allow you to store your projects online, comment, share code and participate in the community forums comfortably [7]. Compiling and programming your projects is really easy. You can literally have the led blink test application running in a few minutes.

Mbed has a dedicated team which develops and maintains a very nice Bluetooth Low Energy API. You can rest assured that using this API for learning to develop BLE *appcessories* will be much easier (and faster) than trying to do so by using directly the BLE libraries (Soft Device) provided by Nordic, because they are already integrated into the Mbed platform The BLE API includes also several application examples.

On the not-so-positive side of Mbed platform, you have zero control over the compiler/linker. This has proven to be a showstopper for our own BLE development. At the time of this writing, within Mbed you are only allowed to develop BLE applications for the nRF51822 along with the S110 Soft Device, which only supports peripheral mode. Hence, you can quickly start to develop applications with Mbed which only require the Peripheral Role. But if you want to develop an application requiring the Central Role, you will have to switch to a full-fledged GCC (or Keil or IAR) C/C++ compiler.

Coding is done in C++, and the compiler provides a programming environment in which to write the code as well as the function of compiling the code into an executable file for the platform.

To set up the platform, the BLE Nano mounted on the MK20 was plugged into a USB port, and the proper platform was chosen from a menu within mbed. Some screenshots of the platform selection screen and the mbed complier can be seen in Fig. 2.

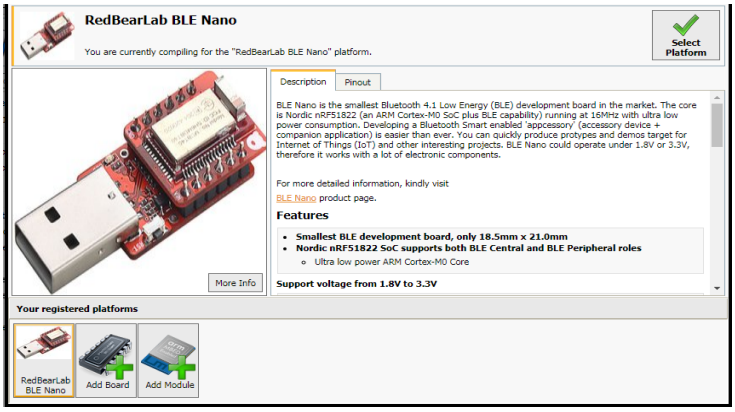

Figure 2. Platform selection screen in mbed, this one showing the RedBearLab BLE Nano

AES algorithm overview:

AES is based on a design principle known as a substitution-permutation network, a combination of both substitution and permutation, and is fast in both software and hardware. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification *per se* is specified with block and key sizes that may be any multiple of 32 bits, with a minimum of 128 and a maximum of 256 bits.

AES Algorithm consists of 2 Main Parts:

1- Encryption or Decryption Process:

In each round, the AES uses the four following operations [8]:

• SubBytes: Each byte of the array is transformed using a nonlinear substitution box called the AES S-Box. The S-Box in the AES has been carefully constructed and the cipher uses only one S-Box throughout the encryption.

• ShiftRows: Is a transposition step which ensures that the last three rows of the array are shifted by a different number of byte positions.

• MixColumns: Mixes each column in the array to create even more diffusion.

• Addkey: Using bitwise XOR, each byte of the array is mixed with a byte of a sub-key material, also called round-key. The sub-key is made by "key expansion" and is derived from the main cipher key using a Rijndael key-schedule.

Encryption process: It starts with AddKey with Key0.

Then go to loop and do SubByte, ShiftRow,MixColumn and AddRoundKey in that order for 9 Rounds each

round with different RoundKey(1-9).

Then go to loop and do SubByte, ShiftRow,MixColumn and AddRoundKey in that order for 9 Rounds each
round with different RoundKey(1-9).
Then go to loop and do SubByte, ShiftRow,MixColumn and AddRoundKey in that order for 9 Rounds each
round with different RoundKey(1-9).
Then go to loop and do SubByte, ShiftRow,MixColumn and AddRoundKey in that order for 9 Rounds each
round with different RoundKey(1-9).

Then go to loop and do SubBytes, ShiftRows, MixColumns, Addkey in that order for 9 circles each circle with different circle key.

Then go to final circle (circle 10) and repeat the same previous function in the loop except MixColumns.

Decryption process: it is reverse of encryption process in every step which means the decryption first circle is the tenth circle of the encryption and it uses the invers functions of MixColumns, SubBytes, ShiftRows and us you can assume the keys arrangement and reversed too as it starts with Addkey10 instead of Addkey0 as it was in the encryption process.

2- Key generation.

It involves RotWord, SubBytes and XOR bitwise operation to generate enough keys for each circle In the Encryption, Decryption process.

As each circle works with different key generated from the key generation process.

AES-CBC buffer encryption and decryption.

Length should be a multiple of the block size (16 bytes). Upon exit, the content of the IV (initialization vector) is updated so that you can call the function same function again on the following block(s) of data and get the same result as if it was encrypted in one call. This allows a "streaming" usage. If on the other hand you need to retain the contents of the IV, you should either save it manually or use the cipher module instead. parameter ctx - the AES context to use for encryption or decryption, parameter mode - MBEDTLS_AES_ENCRYPT or MBEDTLS_AES_DECRYPT, parameter length - length of the input data, parameter iv -initialization vector (updated after use), parameter input-buffer holding the input data, parameter output-buffer holding the output data, return 0 if successful, or MBEDTLS_ERR_AES_INVALID_INPUT_LENGTH.

small_aes.c

```
#include "small_aes.h"

static const unsigned int rcon[] = {
    0x01000000, 0x02000000, 0x04000000, 0x08000000,
    0x10000000, 0x20000000, 0x40000000, 0x80000000,
    0x1B000000, 0x36000000,
};


static const unsigned int Te[256] = {
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,
    0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U,
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,
    0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,
    0x8fcaca45U, 0x1f82829dU, 0x89c9c940U, 0xfa7d7d87U,
    0xeffafa15U, 0xb25959ebU, 0x8e4747c9U, 0xfbf0f00bU,
    0x41adadecU, 0xb3d4d467U, 0x5fa2a2fdU, 0x45afafeaU,
    0x239c9cbfU, 0x53a4a4f7U, 0xe4727296U, 0x9bc0c05bU,
    0x75b7b7c2U, 0xe1fdfd1cU, 0x3d9393aeU, 0x4c26266aU,
    0x6c36365aU, 0x7e3f3f41U, 0xf5f7f702U, 0x83cccc4fU,
    0x6834345cU, 0x51a5a5f4U, 0xd1e5e534U, 0xf9f1f108U,
    0xe2717193U, 0xabd8d873U, 0x62313153U, 0x2a15153fU,
    0x0804040cU, 0x95c7c752U, 0x46232365U, 0x9dc3c35eU,
    0x30181828U, 0x379696a1U, 0x0a05050fU, 0x2f9a9ab5U,
    0x0e070709U, 0x24121236U, 0x1b80809bU, 0xdfe2e23dU,
    0xcdebeb26U, 0x4e272769U, 0x7fb2b2cdU, 0xea75759fU,
    0x1209091bU, 0x1d83839eU, 0x582c2c74U, 0x341a1a2eU,
```

small_aes.h

```
#ifndef _SMALL_AES_H_
#define _SMALL_AES_H_

#include "string.h"

#ifdef __cplusplus
    extern "C" {
#endif

enum {
    SMALL_AES_ENCRYPTION = 0,
    SMALL_AES_DECRYPTION = 1,
    SMALL_AES_BLOCK_SIZE = 16
};

typedef struct AES {
    unsigned int key[60];
    unsigned int  rounds;

    unsigned int reg[SMALL_AES_BLOCK_SIZE / sizeof(unsigned int)];
} AES;
```

terminal output

```
plaintext message: 536f6d6520746869696e6773206172652062657474657220 6c65667420756e7265616400
ciphertext: c57f7afb94f14c7977d785d08682a2596bd62ee9dcf216b8ccd8997afee9b402f5de1739e8e6467aa363749e
decrypted: 536f6d6520746869696e6773206172652062657474657220 6c65667420756e7265616400

DONE
```

Figure 3. Performance of AES ciphers in Mbed platform

AES also has one advantage over other encryption algorithms. Advantage of AES - Cracking a 128 bit AES key with a state-of-the-art supercomputer would take longer than the presumed age of the universe. And Boxcryptor even uses 256 bit keys. As of today, no practicable attack against AES exists. Therefore, AES remains the preferred encryption standard for governments, banks and high security systems around the world.

Security is fundamental for the successful rollout of the Internet of Things. Edge nodes are currently the weakest link in ensuring IoT security and the protection of cryptographic key. The best way to achieve lockdown is by protected hardware. It is the only way to keep those keys and other secrets away from prying eyes. An IoT device can only be as secured as its weakest link.

Homomorphic encryption is a form of encryption which performs arbitrary computations on encrypted data. In cloud computing we may keep our sensitive data in encrypted format, but if you want to do any calculation on cipher text, the key must be shared with cloud service providers which may cause to exploit our data. So that to avoid share the key to cloud service providers instead use the Homomorphic Encryption technique. The computations include searching, sorting, addition, multiplications performed on cipher text.

Among so many cryptographies, homomorphic encryption has attracted widely attentions from scholars for its special performance [9-10]. Common cryptography can't directly do

calculations on encrypted data, but homomorphic encryption can, meanwhile, the operation results of homomorphic encryption will be automatically encrypted. The application prospect of homomorphic encryption is widely and cheerful in the fields of secure multi-party computation, electronic voting, cipher text searching, encrypted mail filtering, mobile cipher. Finally, security analysis is tested and further research way is pointed out. There are two types of homomorphic encryption techniques.

Homomorphic encryption seeks to aid in this encryption process by allowing specific types of computations to be carried out on cipher text which produces an encrypted result which is also in cipher text. Its outcome is the result of operations performed on the plaintext. Case in point, one person could add two encrypted numbers and then another person could decrypt the result, without either of them being able to find the value of the individual numbers.

Partially homomorphic technique, operations are performed on the encrypted data. These operations either additive or multiplicative operation, but not both operation can be carried out at a same time.

In Fully Homomorphic encryption both operations can be carried out at same time. Due to this security mechanism for encrypted data is improved. The first system is a lattice-based encrypted system developed by Craig Gentry in 2009. Fully homomorphic encryption (FHE) is a more powerful technique compared with partially homomorphic encryption. Ideal lattices provide both additive and multiplicative homomorphisms. A cryptosystem that braces arbitrary computation on cipher texts is known as fully homomorphic encryption.

Craig Gentry implemented fully homomorphic encryption based on bootstrapping over partially homomorphic encryption by using ideal lattices. It is limited because each cipher text is noisy in some sense, and this noise grows as one ad and multiplies cipher texts. Gentry showed that any boots trappable Somewhat Homomorphic Encryption scheme can be converted into a Fully Homomorphic Encryption through a self-eMbedding recursion. In case of Gentry's "noisy" scheme, the bootstrapping procedure effectively "refreshes" the cipher text by applying to it the decryption procedure homomorphically, thereby obtaining a new cipher text that encrypts the same value as before but has lower instance of noise [11]. The cipher text is periodically "refreshed" whenever the noise grows too complex.

Fully homomorphic encryption in a binary number ring. The scheme of completely homomorphic encryption, which Gentry suggested, can be considered using the example of calculations in Z_2 [12].

Encryption

The process of data encryption can be represented as follows:

1. We choose an arbitrary odd number $p = 2k + 1$, which is a secret parameter. Let $m \in \{0,1\}$

2. The number $z \in Z_2$ is compiled such that $z = 2r + m$, where r is an arbitrary number. This means that $z = m \bmod 2$.

3. In the process of encryption, each m is associated with the number $c = z + pq$, where q is chosen arbitrarily. Thus, $c = 2r + m + (2k + 1) * q$. It is easy to see that $cmod2 = (m + q)mod2$ and therefore an attacker can determine only the parity of the output of encryption.

Decryption

Let the encrypted number c and the secret p be known. Then the process of decrypting the data should contain the following actions:

1. Decoding using a secret parameter $p: r = cmodp = (z + pq)modp = zmodp + (pq)modp$ where $r = cmodp$ is called noise

2. Obtaining the original encrypted bit: $m = rmod2$.

Justification:

Let there be two bits $m_1, m_2 \in Z_2$ and they are associated with a pair of numbers $z_1 = 2r_1 + m_1$ and $z_2 = 2r_2 + m_2$. Let us take the secret parameter $p = 2k + 1$ and encrypt the data: $c_1 = z_1 + pq_1$ and $c_2 = z_2 + pq_2$.

The sum of these numbers is calculated:

$$c_1 + c_2 = z_1 + pq_1 + z_2 + pq_2 = z_1 + z_2 + p(q_1 + q_2) = 2r_1 + m_1 + 2r_2 + m_2 + (2k + 1)(q_1 + q_2)$$

For the sum of these numbers, the decrypted message is the sum of the original bits $m_1 m_2$: $\big((c_1 + c_2)modp\big)mod2 = (2(r_1 + r_2) + m_1 + m_2)mod2 = m_1 + m_2$. But without knowing p, decoding the data is not possible: $\big((c_1 + c_2)modp\big)mod2 = m_1 + m_2 + q_1 + q_2$.

Similarly, the operation of multiplication is checked:

$$c_1 c_2 = (z_1 pq_1)(z_2 pq_2) = z_1 z_2 + p(z_1 q_2 + z_2 q_1) + p^2 q_1 q_2 = (2r_1 + m_1)(2r_2 + m_2) + (2k + 1)\big((2r_1 + m_1)q_2 + (2r_2 + m_2)q_1\big) = 4r_1 r_2 + 2(r_1 m_2 + r_2 m_1) + m_1 m_2 + 2k(2r_1 q_2 + m_1 q_2 + 2r_2 q_1 + m_2 q_1) + 2r_1 q_2 + m_1 q_2 + 2r_2 q_1 + m_2 q_1$$

The decryption procedure must be applied to the results obtained, which will result in the following: $\big((c_1 c_2)modp\big)mod2 = (4r_1 r_2 + 2(r_1 m_2 + r_2 m_1) + m_1 m_2)mod2 = m_1 m_2$.

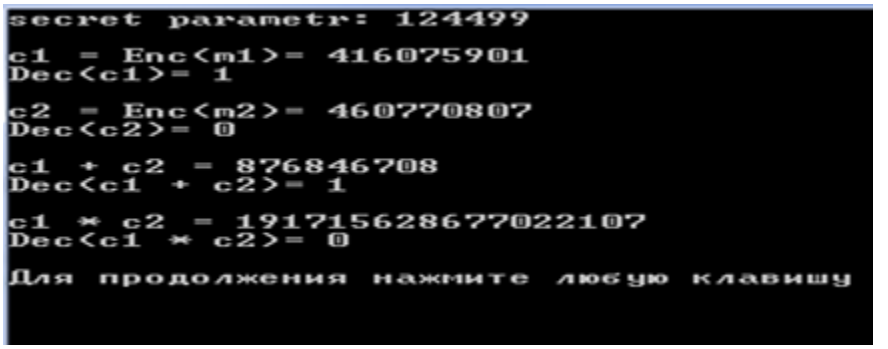

Figure 4. Demonstrated fully homomorphic encryption in a binary number ring in C++

In this paper the implementation AES and fully homomorphic encryption in a binary number ring algorithm. Has given it more encryption power thus makes it harder for anyone to hack the ciphered information and decrypted it.

References

1. BLE Nano. http://redbearlab.com/blenano/.
2. https://www.Mbed.com/en/technologies/security/Mbed-tls/
3. Red Bear Company (2018), http://redbearlab.com/blenano/, Accessed: 18th November, - 2018.
4. Jose Angel, BLE Nano hardware development kit for Bluetooth Low Energy, - 2015
5. S. Aguilar, R. Vidal, C. Gomez, Opportunistic Sensor Data Collection with Bluetooth Low Energy. *Sensors* - 2017, - P. 159.
6. C. Gomez, J. Oller, J. Paradells, Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. Sensors -2012, - P. 11734–11753.
7. http://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.0.pdf
8. J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, C. Gomez, IPv6 over BLUETOOTH(R) Low Energy; RFC 7668; IETF: Fremont, CA, USA, - 2015
9. https://www.mbed.com/en/technologies/security/mbed-tls/
10. EMbedded C++ Technical Committee. “The eMbedded C++ programming guide lines”. http://www.caravan.net/ec2plus/guide.html
11. Atmel Corporation,"Integrating the Internet of Things: Necessary building blocks for broad market adoption", San Jose, USA: Atmel, 0776 Corporate IOT WhitePaper US 102014.
12. Jessica Chani Cahuana, Chalmers, A Search for a Convenient Data Encryption Algorithm For an Internet of Things Device. – 2016. - P. 102-106
13. Alex Biryukov and Dmitry Khovratovich, Related-key Cryptanalysis of the Full AES-192 and AES-256, "Archived copy". Archived *from the original on 2009-09-28.* Retrieved 2018-02-16.
14. Daemen, Joan; Rijmen, Vincent. "AES Proposal: Rijndael". National Institute of Standards and Technology. – 2003. - P. 10-17.
15. N.P. Smart and F. Vercauteren, “Fully homomorphic encryption with relatively small key and ciphertext sizes,” public Key Cryptography-PKC Springer Berlin Heidelberg, vol. 6056, - 2010. –P. 420-443.