https://doi.org/10.31436/iiumej.v21i1.1179

SOFTWARE, ALGORITHMS AND METHODS OF DATA ENCRYPTION BASED ON NATIONAL STANDARDS

RAKHMATILLO DJURAEVICH ALOEV[1], MIRKHON MUKHAMMADOVICH NURULLAEV[2*]

[1]Department of Computational Mathematics and Information Systems, National University of Uzbekistan named after M. Ulugbek, Tashkent, Uzbekistan
[2]Department of Information Communication Technology, Bukhara Engineering Technological Institute, Bukhara, Uzbekistan

**Corresponding author: mirxon@mail.ru*

(Received: 13th June 2019; Accepted: 30th September 2019; Published on-line: 20th January 2020)

ABSTRACT: The article provides a brief description of the cryptography service provider software developed by the authors of this article, which is designed to create encryption keys, create private and public keys of electronic digital signature, create and confirm authenticity of digital signatures, hashing, encrypting, and simulating data using the algorithms described in the State Standards of Uzbekistan. It can be used in telecommunications networks, public information systems, and government corporate information systems by embedding into applications that store, process, and transmit information that does not contain information related to state secrets, as well as in the exchange of information, and ensuring the legal significance of electronic documents. The cryptography service provider includes the following functional components: a dynamically loadable library that implements a biophysical random number sensor; a dynamic library that implements cryptographic algorithms in accordance with the State Standards of Uzbekistan; a module supporting work with external devices; an installation module that provides the installation of a cryptography service provider in the appropriate environment of operation (environment).

ABSTRAK: Artikel ini memberikan penerangan ringkas tentang perisian penyedia perkhidmatan kriptografi yang dibangunkan oleh pengarang artikel ini, yang direka untuk membuat kunci penyulitan, kunci persendirian dan awam tandatangan digital elektronik, membuat dan mengesahkan kesahihan tandatangan digital, hashing, penyulitan dan simulasi data menggunakan algoritma yang dinyatakan dalam Standard Negeri Uzbekistan. Ia boleh digunakan dalam rangkaian telekomunikasi, sistem maklumat awam, sistem maklumat korporat kerajaan dengan memasukkan aplikasi aplikasi yang menyimpan, memproses dan menghantar maklumat yang tidak mengandungi maklumat yang berkaitan dengan rahsia negara, serta pertukaran maklumat dan memastikan undang-undang kepentingan dokumen elektronik.

Penyedia perkhidmatan kriptografi termasuk komponen berfungsi sebagai berikut: perpustakaan dinamik yang boleh dimuatkan yang melaksanakan sensor nombor rawak biofisika; perpustakaan dinamik yang melaksanakan algoritma kriptografi mengikut Standard Negeri Uzbekistan; modul menyokong kerja dengan peranti luaran; modul pemasangan yang menyediakan pemasangan penyedia perkhidmatan kriptografi dalam persekitaran operasi yang sesuai (persekitaran).

KEY WORDS: *Tools of cryptographic protection of information, Data encryption algorithm, Cryptographic provider, Hash function, Encryption key.*

https://doi.org/10.31436/iiumej.v21i1.1179

1. INTRODUCTION

The cryptography service provider (CSP) provides the creation of private and public electronic digital signature keys and encryption keys; creation and confirmation of authenticity of electronic digital signature according to the algorithms described in [2, 7]; the formation of derived encryption keys used by data encryption algorithms described in [4, 5]; work with key information stored on external media; hashing of memory areas and other data according to the algorithms described in [3, 6]; and encryption of memory areas and other data in accordance with the data encryption algorithms described in [4, 5].

The cryptography service provider provides support for identifiers of algorithms and parameters for compatibility with third-party crypto-providers in terms of the ability to work with public-key certificates issued by third-party registration centers, provided they use the cryptographic algorithms described in [2, 3, 4, 5, 6, and 7]. The cryptography service provider provides the ability to work with digital certificates of public keys, which are structured in binary ASN.1 format, conforming to the ITU-T X.509 v.3 standard and IETF RFC 5280 and RFC 3739 recommendations. The cryptography service provider provides work with external key carriers such as USB-flash, and eToken Aladdin (eToken PRO 72K (JAVA)). As part of CIPF[1] – CSP, modules are provided that provide for calling cryptographic functions through the Microsoft CryptoAPI 2.0 interface when running under Microsoft operating systems.

In accordance with the functional purpose of a cryptography service provider, it generates public and private electronic digital signature keys, hash keys, functional keys and encryption keys for use, respectively, in the algorithms described in [2, 3, 4, 5, 6, and 7]. In applications where the cryptography service provider will be integrated, an appropriate key manufacturing and distribution system is provided, which will be based on the key generation functions of the cryptography service provider. The cryptography service provider allows the use of a multi-level key protection model using random and derivative keys of key protection. Protection of keys is carried out on the basis of cryptographic transformations in accordance with the PKCS #5 standard [9] using the State Standards of Uzbekistan [3], [4] or the interstate standard [5]. A cryptography service provider provides storage of key information on a hard magnetic disk drive (HDD) and/or external key storage devices, such as USB-flash or eToken Aladdin (eToken PRO 72K (JAVA)). A cryptography service provider provides work with key containers of signature keys, encryption keys, and additional information necessary to ensure the cryptographic protection of keys and ensure control of their integrity. To protect key information from substitution and/or distortion during its storage on HDD and external key carriers, as well as during distribution, key information is supplied with a checksum. In order to ensure the safe use of the cryptography service provider installed on a PC, organizational measures are provided, as well as software and hardware methods and means of protecting information are used to ensure the secrecy of secret keys located in the PC's memory during the operation of the cryptography service provider, as well as the service cryptography service provider parameters stored on the hard disk. The cryptography service provider contains a component that allows you to verify the operability of the cryptographic algorithms implemented in it. Functioning is carried out on the basis of test examples. To ensure the safe use of an application with a built-in ICS-CSP, mechanisms are provided to control the integrity of the cryptography service provider libraries. In the cryptography service provider, a biophysical sensor of random numbers is used to generate random binary

[1] CIPF - Cryptographic Information Protection Facility

sequences, which implements the mechanism for generating secret digital signature keys, encryption keys, initialization vectors using various algorithms.

The paper briefly describes the software necessary for the operation of the tools of cryptographic protection of information (TCPI) - data encryption cryptography service provider which is developed by the authors based on national standards. Furthermore, the purpose, capabilities of the TCPI - CSP software and its main characteristics, and the limitation of the using area of this software are given here.

One of the following operating systems is necessary in order to perform the TCPI - CSP: Microsoft Windows XP (32 bit) Professional SP3; Microsoft Windows Vista (32 bit) Ultimate SP2; Microsoft Windows 7 (32 bit) Ultimate; Microsoft Windows Server 2003 (32 bit) Enterprise Edition R2 SP2; Microsoft Windows Server 2008 (32 bit) Enterprise Edition SP2.

The original programming languages for the TCPI - CSP are C and C ++.

TCPI - CSP performs the following main functions:

- generation of encryption keys for the data encryption algorithm O`z DSt[2] 1105: 2009 [1] and the algorithm GOST[3] 28147-89 [6];
- encryption of RAM[4] areas and other data in accordance with the data encryption algorithm O`z DSt 1105: 2009 [1] and the algorithm GOST 28147-89 [6];
- generation of keys for implementing and verifying the electronic digital signature (EDS) using algorithms 1 and 2 of O`z DSt 1092: 2009 [3] and the algorithm of GOST R 34.10-2001 [8];
- hashing of memory areas and other data according to algorithm 1 with the parameter p = 256 O`z DSt 1106: 2009 [2] and the algorithm GOST 34.11-94 [7];
- formation and verification of the EDS result in accordance with O`z DSt 1092: 2009 [3] algorithms 1 and 2 and GOST R 34.10-2001 [8];

work with key information stored on external media.

TCPI - CSP can be used as a default crypto-provider for the Windows operating system. TCPI - CSP supports the cryptographic algorithms of the Republic of Uzbekistan and Russia as well as some of the common cryptographic algorithms used in Windows OS, such as RSA, 3DES, SHA-1 etc.

2. FUNCTIONS OF WORKING WITH KEY INFORMATION

TCPI - CSP product works with key information in a key container - storage (Key Container).

Since TCPI – CSP is built in accordance with Microsoft technology, the container contains the following keys:

- ***AT_KEYEXCHANGE*** key used to encrypt and exchange session keys;
- ***AT_SIGNATURE*** - keys used to create and verify a digital signature.

[2] O`z DSt - State standard of Uzbekistan
[3] GOST - Interstate standard
[4] RAM - Random Access Memory

https://doi.org/10.31436/iiumej.v21i1.1179

Note: Private keys (encryption keys and secret signature keys) contained in the container are protected using a security key, which is derived from the value of the user's PIN-code of the token.

Cryptographic procedures are invoked in TCPI - CSP using the PKCS #11 interface.

The underlying concepts of the PKCS #11 interface are slot and token. The token is a repository of some personal information (various keys, certificates, private data, etc.), and the slot acts as a link between a computer and a token that allows different tokens to be connected at different times.

For both ***AT_KEYEXCHANGE*** and ***AT_SIGNATURE***, the same and different PKCS #11 slots can be used.

TCPI - CSP supports work with containers located both on the hard disk of the computer and on removable media such as Flash Memory and Smart Card.

Each container has a unique name consisting of a prefix or several prefixes and the name itself. Prefixes in the container name are separated from each other by the "\" symbol. The container name can contain from zero to three prefixes:

Container Name = [pref1 \] [pref2 \] [pref3 \] Name

The location of the media is determined by the first prefix in the container name, depending on the presence of the ***CRYPT_MACHINE_KEYSET*** flag in the ***CPAcquireContext*** function.

In the container name, the second prefix is a reference to the slot for ***AT_KEYEXCHANGE***, and the third is for ***AT_SIGNATURE***. If the third prefix is absent, then ***AT_KEYEXCHANGE*** and ***AT_SIGNATURE*** are stored in the same slot.

Protection of token's private objects is carried out using the PKCS #5 cryptographic interface. This algorithm solves two problems at once: encrypting private data and protecting it from accidental or intentional distortion.

3. ENCRYPTION FUNCTION

The encryption function is a cryptographic algorithm, which is a bijective mapping from a finite set of plaintext to a finite set of encrypted texts, in which the mapping function depends on a secret parameter called a key. The encryption function is used to encrypt and decrypt information. The encryption function in accordance with [1] can use cryptographic keys of length **256** or **512 bits** for encrypting and decrypting data blocks of length **256 bits**. The encryption function is used for cryptographic protection of data storing and transmitting in computer networks, telecommunications, in separate computing systems or in computers of enterprises, organizations, and institutions.

In symmetric cryptosystems, data exchange takes place in three stages:

1) the sender of the message sends the encryption key (or/and functional key) to the recipient via a secure channel that is known only to them;
2) the sender, using the encryption key and the function key, converts the original data into encrypted data and sends them to the recipient via the communication channel;

3) the recipient, having received the encrypted data, decrypts it with the help of an encryption key and a function key. Both sides may use these keys several times.

In addition to data protection, the encryption function can be used to protect the symmetric keys themselves as they are transmitted over unprotected communication channels. In this case, the transmitted symmetric key is encrypted with some other key, called the security key.

The encryption function [1] contains two modes:

- electronic codebook mode (ECM);
- block chaining mode (BCM).

The electronic codebook mode is an encryption mode in which all plaintext blocks are encrypted independently of one another on the same key, in accordance with the data encryption algorithm.
ECM mode is usually used when encrypting symmetric keys.
The block chaining mode of encryption is a mode in which each encrypted (decrypted) depends on the previous block of an encrypted (decrypted) block. For the first block, the initialization vector is used as the previous block. If the last block of text is not complete, it is supplemented to the required length. This procedure is called padding. BCM mode is usually used when encrypting data.
The purpose of these functions and functionality of operation algorithms are presented in the document "O`z DSt 1105: 2009. State Standard of Uzbekistan. Information technology. CRYPTOGRAPHIC PROTECTION OF INFORMATION. Data encryption algorithm."

4. HASH FUNCTION

The hash function is designed to implement a unidirectional compressing mapping $\boldsymbol{f}$ from set $\boldsymbol{A}$ to set $\boldsymbol{B}$, the input of which is a message of arbitrary length $\boldsymbol{M}$, and the output is a string of fixed length $\boldsymbol{h(M)}$. Using a hashing transform allows you to reduce the input text redundancy.

The hashing function is used in cryptographic methods for processing and protecting information, including for the implementation of electronic digital signature procedures (herein after referred to as EDS[5]) when transferring, processing and storing information in automated systems.

The following are the basic requirements for a hash function:

- the input of the function must be a message of any length;
- at the output of the function, a message of fixed length is obtained;
- the hash function is simply calculated for any message;
- hash function - unidirectional function;
- knowing the message $\boldsymbol{M}$, it is almost impossible to find another message $\boldsymbol{M'}$ for which $\boldsymbol{h(M) = h(M')}$.

In the TCPI - CSP hash function, the output sequence and the hash key have fixed lengths of ***256 bits***.

[5] EDS - electronic digital signature

The composition and purpose of this function, functionality, and functioning algorithm are presented in the document "O`z DSt 1106: 2009. State Standard of Uzbekistan. Information technology. CRYPTOGRAPHIC PROTECTION OF INFORMATION. A hash function".

5. SIGNATURE FUNCTION

The electronic digital signature function is used to generate and confirm the authenticity of an electronic digital signature (EDS) under a given message (electronic document) transmitted over unprotected public telecommunications channels. Upon receipt of the message, the recipient can verify the integrity of the message transmitted by the sender and verify the authenticity of the sender's authorship. EDS is an electronic analog of a written signature and therefore an EDS can be used by the recipient or a third party to verify that the message was actually signed by the sender. To describe the formation and confirmation of the authenticity of a digital signature, two algorithms are used (Algorithm 1, Algorithm 2)[6]. Algorithm 1 is considered in two basic modes:

- without session key[7];
- with a session key.

Algorithm 2 is used in the classical (without session key) mode. Algorithm 1 provides a backup path for detecting a fake digital signature by introducing a session key procedure used in the process of authenticating the authenticity of a digital signature to the EDS generation process.

The composition and purpose of this function, functionality, and functioning algorithm are presented in the document "O`z DSt 1092: 2009. State Standard of Uzbekistan. Information technology. CRYPTOGRAPHIC PROTECTION OF INFORMATION. Processes of formation and verification of electronic digital signature".

Functional restrictions on the use of TCPI – CSP

The cryptographic interface TCPI - CSP is implemented in accordance with the CSP standard, which is applicable only in the Windows operating system and is not applicable in other operating systems such as Linux, Mac, Unix, etc.

Since cryptographic algorithms of the Republic of Uzbekistan are implemented in TCPI - CSP, which are not recognized by standard Windows tools, embedding TCPI in typical Windows applications (MS Outlook, Internet Explorer, VPN, etc.) requires changes to the standard OS software (advapi32.dll, cryptsp.dll, crypt32.dll, inetcomm.dll, schannel.dll, secur32.dll, mailcomm.dll, etc.). Making such changes to Windows can be done in various ways.

6. DESCRIPTION OF THE LOGICAL STRUCTURE

6.1. The algorithms and methods used

When implementing cryptographic algorithms of the Republic of Uzbekistan, including the implementation of operations with big numbers, the source texts of programs from the

[6] Algorithm 1, Algorithm 2 - A description of these algorithms is given in the document "O`z DSt 1092: 2009. State standard of Uzbekistan. Information technology CRYPTOGRAPHIC PROTECTION OF INFORMATION. Processes of formation and verification of electronic digital signature".

[7] a session key is a single-use symmetric key used for encrypting all messages in one communication session.

OpenSSL[8] the software package was used. When implementing cryptographic algorithms of the Republic of Uzbekistan, all simple numbers that are part of the parameters of the algorithms are checked for simplicity using the standard procedures contained in the OpenSSL package.

6.2. Data Encryption Algorithm

Data encryption in TCPI - CSP supports both cryptographic algorithms of the Republic of Uzbekistan [1] and cryptographic standards of the Russian Federation (GOST 28147-89) [6].

Data encryption

Cryptographic provider TCPI - CSP supports various algorithms for symmetric data encryption (SDE) [10], including the data encryption algorithm of the Republic of Uzbekistan [1]. According to section 6.4 [1], an SDE in TCPI - CSP is implemented in three different ways:

- DEA[9] with key 256-bit;
- DEA with key 512-bit;
- DEA with the function key update.

Here is the implementation of all three of these methods using the TCPI - CSP.

6.2.1. DEA with key 256-bit

To implement a DEA with a key of 256 bits, you need to:

1) get the key for the algorithm ***CALG_SYMM***[10]. This key can be obtained in the following ways, in which ***Algid = CALG_SYMM***:
   - CryptGenKey,
   - CryptDeriveKey,

   Either through CryptImportKey from ***SIMPLEBLOB***[11] or ***SYMMETRICWRAPKEYBLOB***[12], created previously via CryptExportKey;
2) call function CryptEncrypt or CryptDecrypt depending on the operation performed.

6.2.2. DEA with key 512-bit

To implement a DEA with a key of 512 bits, you need to:

1) get the key of the DEA algorithm with the key 256 (in accordance with paragraph 1 of the DEA with the key of 256 bits);
2) perform a function CryptSetKeyParam with parameter ***KP_FUNC_KEY*** (#define KP_FUNC_KEY 200). The value ***pbData*** will be the value of the function key that can be generated, in particular, using the function CryptGenRandom or by other means;
3) perform a function CryptEncrypt or CryptDecrypt depending on the operation performed.

[8] **OpenSSL** - OpenSSL is a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library.
[9] **DEA** - Data Encryption Algorithm
[10] ***CALG_SYMM*** - The description of this parameter is given in the document [1]
[11] ***SIMPLEBLOB*** - The description of this parameter is given in the document [1]
[12] ***SYMMETRICWRAPKEYBLOB*** - The description of this parameter is given in the document [1]

6.2.3. DEA with the function key update

To implement a DEA with a function key update, you need to:

1) get the key of the DEA algorithm with the key 256 (in accordance with paragraph 1 of the DEA with the key of 256 bits);
2) if necessary, install a function key (see p. 2 of the DEA with a key of 512 bits);
3) perform a function CryptSetKeyParam with parameter ***KP_OID*** (#define KP_OID 102). As a value for ***pbData*** fed to the input value of the function ***szOID_SYMM_B***[13];
4) call function CryptEncrypt or CryptDecrypt depending on the performed operation.

In the encryption function, the lengths of the input and output blocks, as well as the length of the elements of the array ***Holat***[14] are equal ***256 bit***. The length of the encryption key and the function key are also equal ***256 bit***. The number of steps for the encryption feature is set to $\boldsymbol{e = 8}$.

In both the encryption mode and the decryption mode, the algorithm uses a one-time conversion - forming an array of the session key and the next four bytes - oriented and one bit - oriented conversion at each stage. These transformations include:

- forming arrays of step keys;
- mixing data based on the session key array;
- cyclic shifts of rows and columns of the array ***Holat*** for various values of displacement;
- byte-wise replacement of bytes of the ***Holat*** array based on linear array arrays;
- addition operation modulo 2 ***Holat*** arrays and an array of the stage key $\boldsymbol{K_e}$;
- cyclic shifts of a linear array of a session-stage key by the same value of bits at each stage.

When encrypting a cryptographic module is initialized, the encryption key $\boldsymbol{k}$ and functional key $\boldsymbol{k_f}$, number of stages $\boldsymbol{e}$ and also initialization vector ***IV*** for mode ***m=ShBil*** is first loaded into the cryptographic module. Also, when encrypting a cryptographic module into the ***Holat*** array, the plaintext is loaded; when decrypted, the ciphertext is loaded. At the beginning of the encryption procedure, ***ShaklSeansKalitBayt(k, k_f)***[15], ***ShaklSeansKalit(K_{st})***[16] and ***ShaklBosqichKalit(k_{se})*** crypto-transformations are initialized. At the outputs of crypto transformations ***ShaklSeansKalitBayt(k, k_f), ShaklSeansKalit(K_{st})***, byte-level arrays of substitutions and a session key consisting of diamatrix parts are formed at the byte level. These arrays are used in the following sessions as long as $\boldsymbol{k}$, $\boldsymbol{k_f}$ remain constant. At the output of the crypto-transformation ***ShaklBosqichKalit(k_{se})*** the initial key and the set of stage keys formed for each stage are formed.

7. CRYPTOGRAPHIC TRANSFORMATIONS

Aralash() – is a function that is a cryptographic transformation and is performed on diamatrix parts during encryption and decryption.

[13] ***szOID_SYMM_B*** - The description of this parameter is given in the document [1]

[14] ***Holat*** - array containing one block of information

[15] ***ShaklSeansKalitBayt(k, k_f)*** - is a function that is used to generate the key for each session and to perform the BaytAlmash() conversion when encrypting and decrypting.

[16] ***ShaklSeansKalit(K_{st})*** - is a function that is used to generate the key for each session and to perform the Aralash() transformation when encrypting and decrypting.

When encrypting, the input data are the diamatrix parts of the ***Holat*** array, the $\boldsymbol{K_1}$ and $\boldsymbol{K_2}$ arrays, the output is the ***Holat*** array.

The ***Aralash (Holat, K_s)*** transformation involves performing the following operations:

- if ***m=sh***, then:
  1) is accepted $\boldsymbol{K_1 = K_{1t}, K_2 = K_2}$;
  2) is calculated $\boldsymbol{H_1 ®_2 K_1 (mod\ p), H_2 ®_2 K_2 (mod\ p)}$;
  3) the result is written to arrays $\boldsymbol{H_1, H_2}$;
  4) copy the result to an array ***Holat***;
- if ***m=dsh***, then:
  5) is accepted $\boldsymbol{K_1 = K_1, K_2 = K_{2t}}$;
  6) is calculated $\boldsymbol{H_1 ®_2 K_1 (mod\ p), H_2 ®_2 K_2 (mod\ p)}$;
  7) the result is written to arrays $\boldsymbol{H_1, H_2}$;
  8) copy the result to an array ***Holat***.

The operation of diamatrix multiplication $\boldsymbol{®_2}$ is performed on the basis of the following expressions, here the index used in the expressions takes the values $\boldsymbol{s \in \{1, 2\}}$.

Expressions for $\boldsymbol{i = j \in \{0, 1, 2, 3\}}$:

$$\boldsymbol{h'_s[0,0] = h_s[0,0](k_s[0,0] + k_s[1,0] + k_s[2,0] + k_s[3,0]) - h_s[1,1]k_s[1,0] - h_s[2,2]k_s[2,0] - h_s[3,3]k_s[3,0](mod\ p),}$$

$$\boldsymbol{h'_s[1,1] = h_s[1,1](k_s[0,1] + k_s[1,1] + k_s[2,1] + k_s[3,1]) - h_s[0,0]k_s[0,1] - h_s[2,2]k_s[2,1] - h_s[3,3]k_s[3,1](mod\ p),}$$

$$\boldsymbol{h'_s[2,2] = h_s[2,2](k_s[0,2] + k_s[1,2] + k_s[2,2] + k_s[3,2]) - h_s[0,0]k_s[0,2] - h_s[1,1]k_s[1,2] - h_s[3,3]k_s[3,2](mod\ p),}$$

$$\boldsymbol{h'_s[3,3] = h_s[3,3](k_s[0,3] + k_s[1,3] + k_s[2,3] + k_s[3,3]) - h_s[0,0]k_s[0,3] - h_s[1,1]k_s[1,3] - h_s[2,2]k_s[2,3](mod\ p),}$$

Expressions for $\boldsymbol{i \neq j \in \{0, 1, 2, 3\}}$:

$$\boldsymbol{h'_s[0,1] = h_s[0,1](k_s[0,1] + k_s[1,1] + k_s[2,1] + k_s[3,1]) + (h_s[0,0]+h_s[1,0] + h_s[2,0] + h_s[3,0])k_s[0,1] - h_s[0,2]k_s[2,1] - h_s[0,3]k_s[3,1](mod\ p),}$$

$$\boldsymbol{h'_s[0,2] = h_s[0,2](k_s[0,2] + k_s[1,2] + k_s[2,2] + k_s[3,2]) + (h_s[0,0]+h_s[1,0] + h_s[2,0] + h_s[3,0])k_s[0,2] - h_s[0,1]k_s[1,2] - h_s[0,3]k_s[3,2](mod\ p),}$$

$$\boldsymbol{h'_s[0,3] = h_s[0,3](k_s[0,3] + k_s[1,3] + k_s[2,3] + k_s[3,3]) + (h_s[0,0]+h_s[1,0] + h_s[2,0] + h_s[3,0])k_s[0,3] - h_s[0,1]k_s[1,3] - h_s[0,2]k_s[2,3](mod\ p),}$$

$$\boldsymbol{h'_s[1,0] = h_s[1,0](k_s[0,0] + k_s[1,0] + k_s[2,0] + k_s[3,0]) + (h_s[0,1]+h_s[1,1] + h_s[2,1] + h_s[3,1])k_s[1,0] - h_s[1,2]k_s[2,0] - h_s[1,3]k_s[3,0](mod\ p),}$$

$$\boldsymbol{h'_s[1,2] = h_s[1,2](k_s[0,2] + k_s[1,2] + k_s[2,2] + k_s[3,2]) + (h_s[0,1]+h_s[1,1] + h_s[2,1] + h_s[3,1])k_s[1,2] - h_s[1,0]k_s[0,2] - h_s[1,3]k_s[3,2](mod\ p),}$$

$$\boldsymbol{h'_s[1,3] = h_s[1,3](k_s[0,3] + k_s[1,3] + k_s[2,3] + k_s[3,3]) + (h_s[0,1]+h_s[1,1] + h_s[2,1] + h_s[3,1])k_s[1,3] - h_s[1,0]k_s[0,3] - h_s[1,2]k_s[2,3](mod\ p),}$$

$$\boldsymbol{h'_s[2,0] = h_s[2,0](k_s[0,0] + k_s[1,0] + k_s[2,0] + k_s[3,0]) + (h_s[0,2]+h_s[1,2] + h_s[2,2] + h_s[3,2])k_s[2,0] - h_s[2,1]k_s[1,0] - h_s[2,3]k_s[3,0](mod\ p),}$$

https://doi.org/10.31436/iiumej.v21i1.1179

$$h'_s[2,1] = h_s[2,1](k_s[0,1] + k_s[1,1] + k_s[2,1] + k_s[3,1]) + (h_s[0,2]+h_s[1,2] + h_s[2,2] + h_s[3,2])k_s[2,1] - h_s[2,0]k_s[0,1] - h_s[2,3]k_s[3,1](mod\ p),$$

$$h'_s[2,3] = h_s[2,3](k_s[0,3] + k_s[1,3] + k_s[2,3] + k_s[3,3]) + (h_s[0,2]+h_s[1,2] + h_s[2,2] + h_s[3,2])k_s[2,3] - h_s[2,0]k_s[0,3] - h_s[2,1]k_s[1,3](mod\ p),$$

$$h'_s[3,0] = h_s[3,0](k_s[0,0] + k_s[1,0] + k_s[2,0] + k_s[3,0]) + (h_s[0,3]+h_s[1,3] + h_s[2,3] + h_s[3,3])k_s[3,0] - h_s[3,1]k_s[1,0] - h_s[3,2]k_s[2,0](mod\ p),$$

$$h'_s[3,1] = h_s[3,1](k_s[0,1] + k_s[1,1] + k_s[2,1] + k_s[3,1]) + (h_s[0,3]+h_s[1,3] + h_s[2,3] + h_s[3,3])k_s[3,1] - h_s[3,0]k_s[0,1] - h_s[3,2]k_s[2,1](mod\ p),$$

$$h'_s[3,2] = h_s[3,2](k_s[0,2] + k_s[1,2] + k_s[2,2] + k_s[3,2]) + (h_s[0,3]+h_s[1,3] + h_s[2,3] + h_s[3,3])k_s[3,2] - h_s[3,0]k_s[0,2] - h_s[3,1]k_s[1,2](mod\ p),$$

This transformation is more efficient compared to the matrix transformation. Here, when changing one element in the ***Holat*** source array, depending on the address of the changed element, 6 or 7 elements change.

BaytAlmash() – a function that is a cryptographic transformation and is used to replace elements of the ***Holat*** array with elements of the replacement array at the byte level.

The input data of this crypto-transform is ***Holat*** array, linear array of $B_{sA}[256]$ or $B_{sAD}[256]$ replacements at the byte level, output data is ***Holat*** array at the byte level.

BaytAlmash(Holat, B_a) conversion involves performing the following operations:

1) renaming the ***Holat[8,4]*** array, specified at the byte level, as ***Holatb[8, 4]*** at the byte level;
2) if ***m=sh***, then accepted $B_a[256] = B_{sA}[256]$**;** each element of the ***Holatb[8, 4]*** is replaced with an element of the B_a array located at an address equal to the value of the ***Holatb[8,4]*** array element; the resulting ***Holatb[8, 4]*** array is assigned to the ***Holat[8, 4]*** array specified by the byte level;
3) if ***m=dsh***, then accepted $B_a[256] = B_{sAD}[256]$***;*** replaces each element of the array ***Holatb[8,4]*** element of the array B_a located at the address equal to the value of the element of the array ***Holatb[8, 4]***; the resulting ***Holatb [8,4]*** array is assigned to the ***Holat [8,4]*** array specified by the byte level, here $s \in \{1, 2\}$**.**

Sur() – a function that is used when encrypting and decrypting to thoroughly mix the elements of the ***Holat*** array.

The input data of this transformation is the ***Holat*** array, while the output data is encrypted: Hol ***Holat*** array with columns cyclically shifted downwards and rows cyclically shifted to the right; when decrypted, the output is a ***Holat*** array with columns cyclically shifted upwards and rows cyclically shifted to the left.

The ***Sur (Holat)*** conversion is to perform the following operations:

- if $m = sh$, then first cyclically shift the ***j***-column of the ***Holat*** array to $(j + 1)(mod\ 8)$ bytes down, then shift the ***i***-string of the resulting array to $(i + 1)(mod\ 4)$ bytes right;
- if ***m=dsh***, then first cycle the i - string of the ***Holat*** array by $(i + 1)(mod\ 4)$ bytes left, then shift the ***j***-column of the resulting array by $(j + 1)(mod\ 8)$bytes up. Here, $0 \le i \le 4, 0 \le j \le 8$. →↓↑←

https://doi.org/10.31436/iiumej.v21i1.1179

8. KEY GENERATION

ShaklSeansKalitBayt() – a function that is used to generate a key for each session and to perform the ***BaytAlmash()*** transformation when encrypting and decrypting. In this transformation, the input data is the encryption key $\boldsymbol{k}$ and the function key $\boldsymbol{k_f}$, the output is arrays $\boldsymbol{B_{sA}[256]}$ or $\boldsymbol{B_{sAD}[256]}$ byte level. The ***ShaklSeansKalitBayt(k, k_f)*** transformation is to perform the following operations:

1) Calculation $\boldsymbol{k_{se} = k + k'^{*}(1 + k_f * k)}$ and the remaining ***672 bits*** on the left, here $\boldsymbol{k -}$ is ***192 bits*** $\boldsymbol{k_f}$ on the right.

Note. When generating an encryption key, a set of functional keys that are updated using an encryption key, and forming a $\boldsymbol{k_{se}}$ key based on them, they must be checked based on randomness criteria and ***672 bits*** are allocated from the left side of the resultant k_{se};

2) selection from the right side of $\boldsymbol{k_{se}}$ ***256+64 bits***, from the left 256-bit part of the formation of a linear array of $\boldsymbol{K_{st} = [0, 1, 2, 3, \ldots 31]}$, consisting of byte elements, from the remaining 64-bit part, the formation of a linear array $\boldsymbol{B = [0, 1, 2, 3, 4, 5, 6, 7]}$, consisting of elements at the byte level;

3) formation of a pair of arrays $\boldsymbol{B}$ from the elements of the linear array $\boldsymbol{B_1 = [0, 1, 2, 3]}$ and $\boldsymbol{B_2 = [4, 5, 6, 7]}$ and the formation of the three parameters $\boldsymbol{(d_1, R_1, L_1)}$ and $\boldsymbol{(d_2, R_2, L_2)}$ based on the following rules:

- for $\boldsymbol{j = 0}$, if $\boldsymbol{b[j] < 3}$, is accepted $\boldsymbol{d_1 = 3}$, otherwise – $\boldsymbol{d_1 = b[0]}$;
- for $\boldsymbol{j = 4}$, if $\boldsymbol{b[j] < 3}$, is accepted $\boldsymbol{d_2 = 3}$, otherwise – $\boldsymbol{d_2 = b[4]}$;
- for $\boldsymbol{j = 1}$, if $\boldsymbol{b[j] = 0}$, is accepted $\boldsymbol{R_1 = 1}$, otherwise – $\boldsymbol{R_1 = b[1]}$;
- for $\boldsymbol{j = 5}$, if $\boldsymbol{b[j] = 0}$, is accepted $\boldsymbol{R_2 = 1}$, otherwise – $\boldsymbol{R_2 = b[5]}$;
- for $\boldsymbol{j = 2}$, if $\boldsymbol{b[j] = 0}$, is accepted $\boldsymbol{L_1 = 1}$, otherwise – $\boldsymbol{L_2 = b[2]}$;
- for $\boldsymbol{j = 6}$, if $\boldsymbol{b[j] = 0}$, is accepted $\boldsymbol{L_2 = 1}$, otherwise – $\boldsymbol{L_2 = b[6]}$;
- for $\boldsymbol{d_s(mod\ 2) = 0}$, if $\boldsymbol{d_s(mod\ 4) = 0}$, is accepted $\boldsymbol{d_s = d_s - 1}$, otherwise – $\boldsymbol{d_s = d_s + 1}$, here $\boldsymbol{s \in \{1, 2\}}$;
- for $\boldsymbol{d_s(mod\ 2) = 1}$, if $\boldsymbol{d_s - 1(mod\ 4) = 0}$, then is accepted $\boldsymbol{d_s = d_s - 2}$;
- for $\boldsymbol{j = 3}$, if $\boldsymbol{b[j] = 0}$, is accepted $\boldsymbol{b[j] = 1}$;
- for $\boldsymbol{j = 7}$, if $\boldsymbol{b[j] = 0}$, is accepted $\boldsymbol{b[j] = 1}$;

4) formation of a pair of arrays $\boldsymbol{(B_{1A}[256], B_{2A}[256])}$ to perform transformations at the byte level and get the encrypted text;

5) raising to the power $\boldsymbol{d_s}$ with the parameter $\boldsymbol{R_s}$ modulo ***257*** value $\boldsymbol{\big((i + L_z) mod\ 256\big) + 1}$, corresponding to each address $\boldsymbol{i \in \{0, 1, 2, \ldots 255\}}$, the result is given modulo ***256*** and in each step the current value compared with the value of the previous step and $\boldsymbol{i}$. If the values are equal or if the current value is close $\boldsymbol{(|b_{sA}[i - 1] - b_{sA}[i]| \geq 8)}$ to the previous value, then the current value is replaced with $\boldsymbol{(i - b[3])}$) with $\boldsymbol{s = 1}$ and on $\boldsymbol{(i - b[7])}$ with $\boldsymbol{s = 2}$. Here, $\boldsymbol{s \in \{1, 2\}}$.

Note. In the pseudo-code, in the odd-numbered stages, the $\boldsymbol{B_{1A}[256]}$ is used, and in the even-numbered stages, the $\boldsymbol{B_{2A}[256]}$ is used.

The algorithm of calculations includes the following operations:

1) calculation bsA[i]≡(((i+L) mod 256)+1)| ds (mod 257)(mod 256)

for $\boldsymbol{i = 0..255}$;

2) condition check $\mathbf{i - bsA[i] \neq 0}$ and $\mathbf{|bsA[i-1] - bsA[i]| \geq 8}$, at every step since $\boldsymbol{i = 1}$. If both conditions are met, then the value is accepted. Otherwise, when $\boldsymbol{s = 1}$, swap $\boldsymbol{bsA[i]}$ and the element located at $\boldsymbol{(i - b[3])(mod\ 256)}$, or when $\boldsymbol{s = 2}$ the position $\boldsymbol{bsA[i]}$ and the element located at $\boldsymbol{(i - b[7])(mod\ 256)}$. Then accepted $\boldsymbol{(b[3] = b[3] - 5(mod\ 256)}$ for $\boldsymbol{s = 1}$ or $\boldsymbol{b[7] = b[7] - 5(mod\ 256)}$ for $\boldsymbol{s = 2}$;

3) formation of a pair of linear arrays $\boldsymbol{(B_{1A}[256], B_{2A}[256)}$ of the elements $\boldsymbol{b_{sA}[i, i = 0, 1, 2, ...255]}$. It is necessary to get the cipher text at the byte level, i.e. for use in ***sh*** mode;

4) formation of a pair of linear arrays $\boldsymbol{(B_{1AD}[256], B_{2AD}[256)}$ for decryption at the byte level, i.e. for use in ***dsh*** mode;

5) replacing each element $\boldsymbol{b_{sA}[i]}$ with a value in the linear array $\boldsymbol{B_{sA}[256]}$ with an index equal to its value and positioning the elements of the formed array in increasing order of the address. It is necessary to get the cipher text (***dsh***) at the byte level. Here, $\boldsymbol{s \in \{1, 2\}}$.

Note. In the pseudo-code in the odd-numbered stages, the $\boldsymbol{B_{2AD}[256]}$ array is used, in the even-numbered stages, the $\boldsymbol{B_{1AD}[256]}$ array is used;

ShaklSeansKalit() – is a function that is used to generate a key for each session and to perform the ***Aralash()*** conversion when encrypting and decrypting. The input to this transformation is the $\boldsymbol{K_{st} = [32]}$ array at the byte level; the output is a pair of arrays $\boldsymbol{(K_{1t}, K_2)}$ or $\boldsymbol{(K_1, K_{2t})}$, consisting of diamatrix of a special structure.

The ***ShaklSeansKalit*** ($\boldsymbol{K_{st}}$) transformation is to perform the following operations:

1) the formation of a linear array $\boldsymbol{K_{ss} = [0, 1, 2, ...19]}$, consisting of 20 byte elements from the left of the linear array $\boldsymbol{K_{st} = [0, 1, 2, ...31]}$, consisting of byte elements;

- if $\boldsymbol{k_{ss}[i] = 0}$ for $\boldsymbol{i = 0..9}$, to replace $\boldsymbol{k_{ss}[i]}$ on $\boldsymbol{k_{ss}[i] - 1(mod\ p)}$;
- if $\boldsymbol{k_{ss}[6](mod\ 2) = 0}$, to replace $\boldsymbol{k_{ss}[6]}$ on $\boldsymbol{k_{ss}[6] - 1(mod\ p)}$;
- if $\boldsymbol{k_{ss}[16](mod\ 2) = 0}$, to replace $\boldsymbol{k_{ss}[16]}$ on $\boldsymbol{k_{ss}[16] - 1(mod\ p)}$;
- if $\boldsymbol{k_{ss}[6] + k_{ss}[0] + k_{ss}[8] + k_{ss}[3] + k_{ss}[5](mod\ 2) = 0}$, to replace $\boldsymbol{k_{ss}[8]}$ on $\boldsymbol{k_{ss}[8] - 1(mod\ p)}$;
- if $\boldsymbol{k_{ss}[16] + k_{ss}[10] + k_{ss}[18] + k_{ss}[13] + k_{ss}[15](mod\ 2) = 0}$, to replace $\boldsymbol{k_{ss}[18]}$ on $\boldsymbol{k_{ss}[18] - 1(mod\ p)}$;
- if $\boldsymbol{k_{ss}[6] + k_{ss}[1] + k_{ss}[3] + k_{ss}[9] + k_{ss}[4](mod\ 2) = 0}$, to replace $\boldsymbol{k_{ss}[9]}$ on $\boldsymbol{k_{ss}[9] - 1(mod\ p)}$;
  if $\boldsymbol{k_{ss}[16] + k_{ss}[11] + k_{ss}[13] + k_{ss}[19] + k_{ss}[14](mod\ 2) = 0}$, to replace $\boldsymbol{k_{ss}[19]}$ on $\boldsymbol{k_{ss}[19] - 1(mod\ p)}$;
- if $\boldsymbol{k_{ss}[6] + k_{ss}[2] + k_{ss}[3] + k_{ss}[4] + k_{ss}[7](mod\ 2) = 0}$, to replace $\boldsymbol{k_{ss}[7]}$ on $\boldsymbol{k_{ss}[7] - 1(mod\ p)}$;
- if $\boldsymbol{k_{ss}[16] + k_{ss}[12] + k_{ss}[13] + k_{ss}[14] + k_{ss}[17](mod\ 2) = 0}$, to replace $\boldsymbol{k_{ss}[17]}$ on $\boldsymbol{k_{ss}[17] - 1(mod\ p)}$.

2) formation of two-dimensional arrays $\boldsymbol{K_1[4, 4]}$ and $\boldsymbol{K_2[4, 4]}$ from the elements of the linear array $\boldsymbol{K_{ss}}$ in the following order:

https://doi.org/10.31436/iiumej.v21i1.1179

division of the linear array $\boldsymbol{K_{ss} = [0, 1, 2, \dots 19]}$ into two linear arrays $\boldsymbol{K_{ss1} = [0, 1, 2, \dots 9]}$ and $\boldsymbol{K_{ss2} = [10, 11, 12, \dots 19]}$**.** Each of them is uniquely displayed in an ordered set $\{\boldsymbol{k_{s1}[0,1], k_{s1}[0,2], k_{s1}[0,3], k_{s1}[1,0], k_{s1}[2,0], k_{s1}[2,1], k_{s1}[2,2], k_{s1}[3,0], k_{s1}[3,1], k_{s1}[3,2]}\}$ and
$\{\boldsymbol{k_{s2}[0,1], k_{s2}[0,2], k_{s2}[0,3], k_{s2}[1,0], k_{s2}[2,0], k_{s2}[2,1], k_{s2}[2,2], k_{s2}[3,0], k_{s2}[3,1], k_{s2}[3,2]}\}$.
The formation of the elements $\boldsymbol{k_1[i,j], k_2[i,j]}$ of arrays $\boldsymbol{K_1[4,4]}$ and $\boldsymbol{K_2[4,4]}$;

the formation of the remaining elements of the array $\boldsymbol{K_s[4,4],\ s \in \{1,2\}}$ based on the following rule:

- for $\boldsymbol{j \in \{0,1,2,3\}}$, if $\boldsymbol{i = j}$, then the corresponding elements are equal in value $\boldsymbol{k_s[2,2]}$;
- for $\boldsymbol{i = 1, j = 0, 2, 3}$ corresponding elements are equal in value $\boldsymbol{k_s[1,0]}$;
- for $\boldsymbol{i = 2,\ j = 0, 3}$ corresponding elements are equal in value $\boldsymbol{k_s[2,0]}$;

3) as a result, for use in encryption mode as $\boldsymbol{K_s[8,4]}$ a pair of diamatrix of a special structure is formed $\boldsymbol{K_1[4,4]}$ and $\boldsymbol{K_2[4,4]}$***;***

Array $\boldsymbol{K_1}$

$\boldsymbol{k_{ss}[6]}$	$\boldsymbol{k_{ss}[0]}$	$\boldsymbol{k_{ss}[1]}$	$\boldsymbol{k_{ss}[2]}$
$\boldsymbol{k_{ss}[3]}$	$\boldsymbol{k_{ss}[6]}$	$\boldsymbol{k_{ss}[3]}$	$\boldsymbol{k_{ss}[3]}$
$\boldsymbol{k_{ss}[4]}$	$\boldsymbol{k_{ss}[5]}$	$\boldsymbol{k_{ss}[6]}$	$\boldsymbol{k_{ss}[4]}$
$\boldsymbol{k_{ss}[7]}$	$\boldsymbol{k_{ss}[8]}$	$\boldsymbol{k_{ss}[9]}$	$\boldsymbol{k_{ss}[6]}$

$\boldsymbol{k_1[0,0]}$	$\boldsymbol{k_1[0,1]}$	$\boldsymbol{k_1[0,2]}$	$\boldsymbol{k_1[0,3]}$
$\boldsymbol{k_1[1,0]}$	$\boldsymbol{k_1[1,1]}$	$\boldsymbol{k_1[1,2]}$	$\boldsymbol{k_1[1,3]}$
$\boldsymbol{k_1[2,0]}$	$\boldsymbol{k_1[2,1]}$	$\boldsymbol{k_1[2,2]}$	$\boldsymbol{k_1[2,3]}$
$\boldsymbol{k_1[3,0]}$	$\boldsymbol{k_1[3,1]}$	$\boldsymbol{k_1[3,2]}$	$\boldsymbol{k_1[3,3]}$

Array $\boldsymbol{K_2}$

$\boldsymbol{k_{ss}[16]}$	$\boldsymbol{k_{ss}[10]}$	$\boldsymbol{k_{ss}[11]}$	$\boldsymbol{k_{ss}[12]}$
$\boldsymbol{k_{ss}[13]}$	$\boldsymbol{k_{ss}[16]}$	$\boldsymbol{k_{ss}[13]}$	$\boldsymbol{k_{ss}[13]}$
$\boldsymbol{k_{ss}[14]}$	$\boldsymbol{k_{ss}[15]}$	$\boldsymbol{k_{ss}[16]}$	$\boldsymbol{k_{ss}[14]}$
$\boldsymbol{k_{ss}[17]}$	$\boldsymbol{k_{ss}[18]}$	$\boldsymbol{k_{ss}[19]}$	$\boldsymbol{k_{ss}[16]}$

$\boldsymbol{k_2[0,0]}$	$\boldsymbol{k_2[0,1]}$	$\boldsymbol{k_2[0,2]}$	$\boldsymbol{k_2[0,3]}$

https://doi.org/10.31436/iiumej.v21i1.1179

$k_2[1,0]$	$k_2[1,1]$	$k_2[1,2]$	$k_2[1,3]$
$k_2[2,0]$	$k_2[2,1]$	$k_2[2,2]$	$k_2[2,3]$
$k_2[3,0]$	$k_2[3,1]$	$k_2[3,2]$	$k_2[3,3]$

4) calculation of the inverse matrix of a special structure $K_{1t}[4,4]$ for the matrix of the special structure $K_1[4,4]$ for use in ***sh*** mode;
5) calculation of the inverse matrix of the special structure $K_{2t}[4,4]$ for the matrix of the special structure $K_2[4,4]$ for use in ***dsh*** mode;
6) obtaining an inverse matrix for the diamatrix of a special structure $K_{1i}[4,4]$ (here $i=\{1,2\}$), in which the diadetergent is not zero, consists in calculating the inverse matrix for the matrix obtained as a result of performing a diagonal transformation over it and the result of performing a transform over the obtained inverse matrix;
7) as a result of the inversion of the diamatrices of the special structure $K_1[4,4]$ and $K_2[4,4]$ the diamatrices of the special structure $K_{1t}[4,4]$ and $K_{2t}[4,4]$, are formed, having the following form (K_{1t} and K_{2t}).

Array K_{1t}

$k_{1t}[0,0]$	$k_{1t}[0,1]$	$k_{1t}[0,2]$	$k_{1t}[0,3]$
$k_{1t}[1,0]$	$k_{1t}[1,1]$	$k_{1t}[1,2]$	$k_{1t}[1,3]$
$k_{1t}[2,0]$	$k_{1t}[2,1]$	$k_{1t}[2,2]$	$k_{1t}[2,3]$
$k_{1t}[3,0]$	$k_{1t}[3,1]$	$k_{1t}[3,2]$	$k_{1t}[3,3]$

Array K_{2t}

$k_{2t}[0,0]$	$k_{2t}[0,1]$	$k_{2t}[0,2]$	$k_{2t}[0,3]$
$k_{2t}[1,0]$	$k_{2t}[1,1]$	$k_{2t}[1,2]$	$k_{2t}[1,3]$
$k_{2t}[2,0]$	$k_{2t}[2,1]$	$k_{2t}[2,2]$	$k_{2t}[2,3]$
$k_{2t}[3,0]$	$k_{2t}[3,1]$	$k_{2t}[3,2]$	$k_{2t}[3,3]$

In ***sh*** mode, a pair consisting of ***(K_{1t}, K_2)***, is fed to the input; in ***dsh*** mode, a pair consisting of ***(K_1, K_{2t})*** is fed to the input.

ShaklBosqichKalit() – is a function that is used to form a stage key from a session-stage key and to perform the ***Qo'shBosqichKalit()*** transformation when encrypting and decrypting.

The input data of this transformation is the linear array of the session-stage key k_{se}, the output data is the two-dimensional array specified by the byte level $K_e[8,4]$;

The ***ShaklBosqichKalit*** (k_{se}) transformation (generation of a linear session-stage key) occurs as follows:

1) if ***bosqich=1*** and ***m=sh***, then the k_{se} array of the linear session-stage key is left unchanged, if ***bosqich=1*** and ***m=dsh***, then the k_{se} array is shifted by ***672-(e x 83) mod 672 bits*** to the right;
2) the left ***256 bit*** part of the linear session-stage key array is separated and the $K_e[8,4]$ array is formed from it at the byte level. This conversion is performed for all stages prior to the start of the encryption procedure;
3) if $bosqich > 1$ and ***m=sh***, then the k_{se} array cyclically shifts ***83 bits*** to the left, if $bosqich > 1$ and ***m=dsh***, then the k_{se} array cyclically shifts ***83 bits*** to the right;
4) the left ***256 bit*** part of the linear session-stage key array is separated and an array is formed from it at the byte level $K_e[8,4]$. This conversion is performed for all stages prior to the start of the decryption procedure.

9. SIMPLE CRYPTO-TRANSFORMATION

Qo'shBosqichKalit() – is a function that is simple crypto-transform and consists of performing an exclusive "or" (bitwise addition modulo 2) when encrypting and decrypting ***Holat*** arrays and an array of the step-key K_e. The input data of this transformation are the ***Holat*** and K_e arrays at the byte level, the output data is the ***Holat*** array at the byte level. The ***Qo'shBosqichKalit (Holat, K_e)*** transformation consists of performing an exclusive "or" operation (bitwise modulo-2 addition) on elements of the same name at the byte level of ***Holat*** and $K_e[8,4]$ arrays.

For $0 \le c < 8$:

$$[h'[c,0], h'[c,1], h'[c,2], h'[c,3]] = [h[c,0], h[c,1], h[c,2], h[c,3]] \oplus [k_e[c,0], k_e[c,1], k_e[c,2], k_e[c,3]].$$

The result must be copied to the ***Holat*** array.

Qo'shHolat() – is a function that is a simple crypto-transform and is performed on blocks of encrypted blocks using the ***XOR*** operation when encrypting and decrypting in all modes except the electronic codebook mode.

In the ***Qo'shHolat(Holatn, Holat)*** transformation, each byte of the ***Holatn*** array is added bitwise using the ***XOR*** addition operation (modulo 2 addition operation) the same byte of the ***Holat*** array. ***Holatn*** array consists of eight words. When these words are in the range of $0 \le s < 8$, the elements of the ***Holat*** array that are in the columns are added separately as follows:

$$[h'[s,0], h'[s,1], h'[s,2], h'[s,3]] = [h[s,0], h[s,1], h[s,2], h[s,3]] \oplus [hn[s,0], hn[s,1], hn[s,2], hn[s,3]],$$

here: ***hn*** - array elements ***Holatn***, h' – elements of the resulting array.

The result of the conversion is copied to the ***Holat*** array.

9.1. Encryption of symmetric keys using symmetric keys

To encrypt symmetric keys using symmetric keys, a DEA is used in ECM mode. To produce imitation protection, a hash function from cryptographic standards of Uzbekistan in

256 bit mode is used. The first 4 bytes of the key's hash function is the simulated prefix for the encrypted key.

9.2. Encryption of symmetric keys using asymmetric keys

When encrypting symmetric keys using asymmetric keys, the following algorithm is used. Using the Diffie-Hellman method, a common key of the form $\boldsymbol{a^{xy}}$ is formed, where $\boldsymbol{x, y}$ are private keys, the common key size is 256 and 64 bytes for algorithms 1 and 2, respectively. This shared key is used instead of a password to generate a shared symmetric key and initialization vector in accordance with PKCS #5. The original symmetric key is encrypted in BCM mode using a common symmetric key and an initialization vector. The size of the encrypted key is 64 bytes. Imitation protection provides padding size of 32 bytes.

9.3. An algorithm of generation and verification of EDS

Processes of formation and verification of electronic digital signatures in TCPI - CSP supports both cryptographic algorithms of the Republic of Uzbekistan [3] and the cryptographic standard of the Russian Federation (GOST R 34.10-2001) [8]. The function of EDS (formation and verification) is designed to ensure the reliability of the transmitted and received information and confirm its authorship.

9.4. Development and verification of EDS

The following parameters are used for the signature function:

- $\boldsymbol{(x, u)}$ –pair of integers - private key of EDS;
- $\boldsymbol{(y, z)}$ – pair of integers – public key EDS;
- $\boldsymbol{(r, s)}$ – pair of integers – electronic digital signature under the message $\boldsymbol{M}$;
- $\boldsymbol{(R_1, y_1)}$ – pair of integers – fake detection key EDS, which is a pair of control and session public key;
- $\boldsymbol{(R, g, k_h)}$ – special private key of the authorized subject.

The generally accepted digital signature model spans three processes:

- generation of EDS keys;
- formation of EDS;
- verification (authentication) EDS.

9.4.1. Algorithm 1

Algorithm 1 uses the following parameters:

- $\boldsymbol{p}$ – is a module, a prime number. The upper limit of this number should be determined by the specific implementation of the electronic digital signature algorithm, depending on the type of cryptographic module:
- $\boldsymbol{p > 2^{1023}}$ for software, hybrid and hardware types and $\boldsymbol{p > 2^{255}}$ for special hardware type;
- $\boldsymbol{q}$- is a prime number, which is a factor (simple factor) $\boldsymbol{p - 1}$, where $\boldsymbol{2^{25} < q < 2^{256}}$;
- $\boldsymbol{R}$ – is the parameter - a positive integer satisfying the condition $\boldsymbol{R < q}$; $\boldsymbol{R}$ can be an open, shared private key for a limited group of users or a component of a special private key of an authorized entity;
- **m = H($\boldsymbol{M}$)** – is a hash function that displays the message $\boldsymbol{M}$ in a string of length **256 bits**; In software, hybrid and hardware types of a cryptographic module, a hash function without a key is used, and in a special hardware type, a key hash function.

To implement EDS processes (i.e., EDS key generation, EDS generation, and EDS authentication), each user must have a private EDS key (x, u), where: x, u are private keys, randomly or pseudo-randomly generated integers satisfying the condition $1 < x, u < q$; the g parameter is a private or public parameter representing an integer that is calculated: $g \equiv h^{\backslash(p-1)/q}(mod\ p)$, the public key of the digital signature (y, z). Where: (y, z) are public keys calculated by the formula $y \equiv g^{\backslash x}(mod\ p)$ and $z \equiv g^{\backslash u}(mod\ p)$; if the open parameter (base) g is used, then $u = 1$ and $z = g$; and the EDS fake detection key (R_1, y_1), where: R_1 is the control key (open or closed), selected from the range $1..q-1$; if R_1 is closed, then R_1 must be a joint secret key for the signer and the verifier;

- y_1 – is a session (public) key calculated for each electronic digital signature as a result of exponentiation with the parameter.

Formation of electronic digital signature and session key

To create a digital signature and session key under the message **M** for **Algorithm 1**, the following actions are performed**:**

1) the hash function of the message is calculated: $m = H(M)$. Moreover, $c = x$;
2) $k = H(m + (1 + mR)c)$ is calculated. If $k = 0$, then $c = c + 2$ is assumed and it is necessary to return to step 2;
3) calculate $T \equiv g^{\backslash -k}(mod\ p)$ with the parameter R;
4) calculate $r \equiv m + (1 + mR)T(mod\ p)$. If $r\ (mod\ q) = 0$, then $k \equiv k + 1(mod\ p))$ is assumed and it is necessary to return to step 3.
5) $s_1 \equiv k - rx(mod\ q)$ is calculated. If $s_1 = 0$, then $k \equiv k + 1(mod\ p)$ is assumed and it is necessary to return to step 3;
6) compute $s \equiv s_1 u^{-1}(mod\ q)$. If $\mu = 0$, then r, s is output and calculations stop;
7) calculate $r_1 \equiv R_1 + (1 + RR_1)r(mod\ q)$. If $r_1 = 0$, then $k \equiv k + 1(mod\ p)$ is assumed and it is necessary to return to step 3;
8) $x_1 \equiv (k - suR_1){r_1}^{-1}(mod\ q)$ is calculated. If $x_1 = 0$, then $k \equiv k + 1(mod\ p)$ is assumed and it is necessary to return to step 3;
9) $y_1 \equiv (g{R_1}^{-1})^{\backslash x_1}(mod\ p)$ with the parameter RR_1 is calculated and output r, s, y_1.

Further, the signed message (message and addition) is transmitted to the receiving side. Also, the session key is transmitted to the receiving side if the session key mode is used.

Digital Signature Authentication

To confirm the authenticity of EDS under the received message, M, the following actions are performed:

1) hash function $m = H(M)$ is calculated;
2) if $L(s) <= L(q)\ AND\ L(r) <= L(p)$ then go to the next step, otherwise the "signature is not authentic" is accepted;
3) $z_0 \equiv z^{\backslash s}(mod\ p)$ is calculated with the parameter **R;**
4) $r' \equiv r(mod\ q)$ is calculated**;**
5) $y_2 \equiv y^{\backslash r'}(mod\ p)$ is calculated with the parameter **R;**
6) $z_1 \equiv z_0 + (1 + z_0 R)y_2(mod\ p)$ is calculated;
7) $y_3 \equiv z_1 + (1 + z_1 R)r(mod\ p)$ is calculated;

https://doi.org/10.31436/iiumej.v21i1.1179

8) if $\boldsymbol{\mu = 0}$ and $\boldsymbol{m = y_3}$, then the output is "signature authentic";
if $\boldsymbol{\mu = 1}$ and $\boldsymbol{m = y_3}$, then go to the next step;

if $\boldsymbol{m \neq y_3}$, then the "signature is not authentic" is accepted;

9) $\boldsymbol{g_3 \equiv z_1 R_1^{-1} (mod\ p)}$ is calculated;
10) $\boldsymbol{s_1 \equiv s R_1 (mod\ q)}$ is calculated;
11) $\boldsymbol{r_1 \equiv R_1 + (1 + R R_1) r' (mod\ q)}$ is calculated;
12) $\boldsymbol{z_2 \equiv z R_1^{-1} (mod\ p)}$ is calculated;
13) $\boldsymbol{y_4 \equiv y_1}$;
14) $\boldsymbol{z_3 \equiv z_2^{\backslash s_1} (mod\ p)}$ is calculated with the parameter $\boldsymbol{RR_1}$;
15) $\boldsymbol{y_5 \equiv y_4^{\backslash r_1} (mod\ p)}$ is calculated with the parameter $\boldsymbol{RR_1}$;
16) $\boldsymbol{g_4 \equiv z_3 + (1 + z_3 R R_1) y_5 (mod\ p)}$ is calculated;
17) if $\boldsymbol{g_3 \equiv g_4}$, then "signature is authentic" is accepted, otherwise "signature is not authentic" is accepted.

Algorithm 1 uses a **one-way function** in a group with a parameter, calculations for which are carried out easily at the same level of labor intensity as in the exponentiation operations, and inverting (inversion) of a function requires no less computational time and effort than in solving a discrete problem logarithm The main operations of multiplication, exponentiation, and treatment in a group with a parameter are called multiplication, exponentiation, and treatment of the parameter. The one-sided exponentiation function is a special case of this one-way function.

9.4.2. Algorithm 2

For Algorithm 2, the following parameters are used:

- elliptic curve **E**, given by its invariant **J(E)** or by coefficients $\boldsymbol{a, b \in F_p}$;
- integer **w** is the order of a group of points of an elliptic curve **E**;
- prime **t** is the order of a cyclic subgroup of a group of points of an elliptic curve **E** for which the following conditions are satisfied:

$$\begin{cases} \boldsymbol{w = lt, l \in Z, l \geq 1} \\ \boldsymbol{2^{254} < t < 2^{256}} \end{cases}$$

- point $\boldsymbol{N \neq 0}$ of an elliptic curve $\boldsymbol{E}$, with coordinates $\boldsymbol{(x_p, y_p)}$, satisfying the equality $\boldsymbol{[t]N = 0}$;
- hash function $\boldsymbol{m = H(M)}$, displaying the message $\boldsymbol{M}$, in a string of length **256 bits**.

For the listed parameters of EDS the following requirements are met:

1) $\boldsymbol{p^i \neq 1 (mod\ t)}$ for all integers $\boldsymbol{i = 1, 2, \dots, B}$, where $\boldsymbol{B}$ satisfies the inequality $\boldsymbol{B \geq 31}$;
2) $\boldsymbol{w \neq p}$;
3) the invariant of the curve satisfies the condition $\boldsymbol{J(E) \neq 0}$ or **1728.**

To implement EDS processes (i.e., EDS key generation, EDS generation and EDS authentication), each user must have an EDS private key, an integer $\boldsymbol{d}$, satisfying $\boldsymbol{0 < d < t}$

https://doi.org/10.31436/iiumej.v21i1.1179

and the public EDS key is a point of an elliptic curve $\boldsymbol{T}$ with coordinates $\boldsymbol{(x_t, y_t)}$, satisfying the equality $\boldsymbol{[d]N=T\ T(x_t, y_t)}$.

Formation of electronic digital signature

To obtain a digital signature under the message, $\boldsymbol{M}$, the following actions are performed:

1) calculate the hash function of message: $\boldsymbol{m = H(M)}$;
2) calculate $\boldsymbol{e \equiv m(mod\ t)}$. If $\boldsymbol{e = 0}$, then define $\boldsymbol{e = 1}$;
3) a random (pseudo-random) integer $\boldsymbol{k}$ is generated, satisfying the inequality $\boldsymbol{0 < k < t}$;
4) the point of the elliptic curve $\boldsymbol{C = [k]N}$ is calculated and $\boldsymbol{r = x_c(mod\ t)}$, is determined, where $\boldsymbol{x_c}$ is the $\boldsymbol{x}$ coordinate of point $\boldsymbol{C}$. If $\boldsymbol{r = 0}$, then it is necessary to return to step 3;
5) compute the value of $\boldsymbol{s \equiv (rd + ke)(mod\ t)}$. If $\boldsymbol{s = 0}$, then you need to return to step 3;
6) issue to the output of $\boldsymbol{r}$ and $\boldsymbol{s}$ as a digital signature.

Digital Signature Authentication

To confirm the authenticity of EDS under the received message, $\boldsymbol{M}$, the following actions are performed:

1) if $\boldsymbol{0 < r < t, 0 < s < t}$, then go to the next step, otherwise the "signature is not authentic" is accepted;
2) calculate the message hash function: $\boldsymbol{m = H(M)}$;
3) compute by $\boldsymbol{e \equiv m(mod\ t)}$. If $\boldsymbol{e = 0}$, then define $\boldsymbol{e = 1}$;
4) calculate the value of $\boldsymbol{v \equiv e^{-1}(mod\ t)}$;
5) $\boldsymbol{z_1 \equiv sv(mod\ t)}$, $\boldsymbol{z_2 \equiv -rv(mod\ t)}$ values are calculated;
6) the point of the elliptic curve $\boldsymbol{C = [z_1]N" + "[z_2]T}$ is calculated and $\boldsymbol{R \equiv x_c(mod\ t)}$ is defined, where $\boldsymbol{x_c}$ – is the $\boldsymbol{x}$ coordinate of point $\boldsymbol{C}$;
7) if the equality $\boldsymbol{R = r}$ holds, then the "signature is authentic" is accepted, otherwise "the signature is not authentic".

Algorithm of generation and verification of EDS using the algorithm of the Russian Federation

For the algorithm of the Russian Federation, the following parameters of the digital signature scheme are used:

- a prime number $\boldsymbol{p}$ - is a module of an elliptic curve such that $\boldsymbol{p > 2^{255}}$;
- the elliptic curve $\boldsymbol{E}$ is defined by its invariant $\boldsymbol{J(E)}$ or the coefficients $\boldsymbol{a, b \in F_p}$, where $\boldsymbol{F_p}$ is a finite field of p elements. $\boldsymbol{J(E)}$ is related to the coefficients $\boldsymbol{a}$ and $\boldsymbol{b}$ as follows

  $\boldsymbol{J(E) = 1728 * (\frac{4a^3}{4a^3+27b^2})(mod\ p)}$, and $\boldsymbol{4a^3 + 27b^2 \neq 0\ (mod\ p)}$;
- integer $\boldsymbol{m}$ - is the order of a group of points of an elliptic curve, $\boldsymbol{m}$ must be different from $\boldsymbol{p}$;
- a prime number $\boldsymbol{q}$, the order of a cyclic subgroup of a group of points of an elliptic curve, that is, $\boldsymbol{m = n * q}$; for some $\boldsymbol{n \in N}$. Also $\boldsymbol{q}$ lies within $\boldsymbol{2^{254} < q < 2^{256}}$;

- the point $\boldsymbol{P} = (\boldsymbol{x_p}, \boldsymbol{y_p})$ of the elliptic curve $\boldsymbol{E}$, which is the generator of a subgroup of order $\boldsymbol{q}$, that is, $\boldsymbol{q} * \boldsymbol{P} = \boldsymbol{O}$ and $\boldsymbol{k} * \boldsymbol{P} \neq \boldsymbol{O}$ for all $\boldsymbol{k} = \boldsymbol{1}, \boldsymbol{2}, \ldots, \boldsymbol{q} - \boldsymbol{1}$, where $\boldsymbol{O}$ is a neutral element groups of points of an elliptic curve $\boldsymbol{E}$;
- $\boldsymbol{h}(\boldsymbol{M})$ - hash function (GOST R 34.11-94) [7], which displays message $\boldsymbol{M}$ in binary vector length of 256 bits.

Every digital signature user has private keys:

- encryption key $\boldsymbol{d}$ – integer number within $\boldsymbol{0} < \boldsymbol{d} < \boldsymbol{q}$;
- decryption key $\boldsymbol{Q} = (\boldsymbol{x_Q}, \boldsymbol{y_Q})$, calculated as $\boldsymbol{Q} = \boldsymbol{d} * \boldsymbol{P}$.

Additional requirements:

- $\boldsymbol{p^t} \neq \boldsymbol{1}(\boldsymbol{mod}\ \boldsymbol{q})$, any $\boldsymbol{t} = \boldsymbol{1}..\boldsymbol{B}$, where $\boldsymbol{B} \geq \boldsymbol{31}$;
- $\boldsymbol{J}(\boldsymbol{E}) \neq \boldsymbol{1728}$.

Digital Signature Generation

To obtain a digital signature under the message M, the following actions are performed:

1) calculation of the hash function from the message $\boldsymbol{M}$: ĥ $= \boldsymbol{h}(\boldsymbol{M})$;
2) calculation of $\boldsymbol{e} = \boldsymbol{z}\ \boldsymbol{mod}\ \boldsymbol{q}$, and if $\boldsymbol{e} = \boldsymbol{0}$, set $\boldsymbol{e} = \boldsymbol{1}$. Where $\boldsymbol{z}$ is an integer corresponding to ĥ;
3) generation of a random number $\boldsymbol{k}$ such that $\boldsymbol{0} < \boldsymbol{k} < \boldsymbol{q}$;
4) calculation of the point of the elliptic curve $\boldsymbol{C} = \boldsymbol{k} * \boldsymbol{P}$, and finding $\boldsymbol{r} = \boldsymbol{xc}\ \boldsymbol{mod}\ \boldsymbol{q}$ where $\boldsymbol{xc}$ is the $\boldsymbol{x}$ coordinate of the point $\boldsymbol{C}$. If $\boldsymbol{r} = \boldsymbol{0}$, we return to the previous step;
5) verification of $\boldsymbol{s} = (\boldsymbol{r} * \boldsymbol{d} + \boldsymbol{k} * \boldsymbol{e})\ \boldsymbol{mod}\ \boldsymbol{q}$. If $\boldsymbol{s} = \boldsymbol{0}$, go back to step 3;
6) digital signature generation $\boldsymbol{\xi} =$ (ř|ŝ), where ř and ŝ are vectors corresponding to $\boldsymbol{r}$ and $\boldsymbol{s}$.

Verifying Digital Signatures.

To confirm the authenticity of EDS under the received message M, the following actions are performed:

1) calculation by digital signature $\boldsymbol{\xi}$ of numbers r and s, taking into account that $\boldsymbol{\xi} =$ (ř|ŝ), where $\boldsymbol{r}$ and $\boldsymbol{s}$ are numbers corresponding to vectors ř and ŝ. If at least one of the inequalities $\boldsymbol{r} < \boldsymbol{q}$ and $\boldsymbol{s} < \boldsymbol{q}$ is incorrect, then wrong signature;
2) calculation of the hash function from the message $\boldsymbol{M}$: ĥ $= \boldsymbol{h}(\boldsymbol{M})$;
3) calculation of $\boldsymbol{e} = \boldsymbol{z}\ \boldsymbol{mod}\ \boldsymbol{q}$, and if $\boldsymbol{e} = \boldsymbol{0}$, set $\boldsymbol{e} = \boldsymbol{1}$. Where $\boldsymbol{z}$ is an integer corresponding to ĥ;
4) calculation of $\boldsymbol{\vartheta} = \boldsymbol{e} - \boldsymbol{1}\ \boldsymbol{mod}\ \boldsymbol{q}$;
5) calculation of $\boldsymbol{z_1} = \boldsymbol{s} * \boldsymbol{\vartheta}\ \boldsymbol{mod}\ \boldsymbol{q}$ and $\boldsymbol{z_2} = -\boldsymbol{r} * \boldsymbol{\vartheta}\ \boldsymbol{mod}\ \boldsymbol{q}$;
6) calculation of the point of the elliptic curve $\boldsymbol{C} = \boldsymbol{z_1} * \boldsymbol{P} + \boldsymbol{z_2} * \boldsymbol{Q}$;
7) definition $\boldsymbol{R} = \boldsymbol{xc}\ \boldsymbol{mod}\ \boldsymbol{q}$, where $\boldsymbol{xc}$ is the $\boldsymbol{x}$ coordinate of the curve $\boldsymbol{C}$ definition $\boldsymbol{R} = \boldsymbol{xc}\ \boldsymbol{mod}\ \boldsymbol{q}$, where $\boldsymbol{xc}$ – is the $\boldsymbol{x}$ coordinate of the curve $\boldsymbol{C}$;
8) in the case of equality $\boldsymbol{R} = \boldsymbol{r}$ the signature is correct, otherwise - is incorrect.

Program structure

This subsection presents the general structure of the TCPI - CSP, as well as a description of the functions of each module of the system.

10. GENERAL STRUCTURE

TCPI - CSP is implemented in the form of the following dynamic libraries:

- ***CSP.DLL*** - loading the CSP interface using the Crypto API;
- ***CSPFUNC.DLL*** - loading the CSP interface directly;
- ***PKCS11.DLL*** - loading the PKCS #11[17] interface (PKCS #11 interface for TCPI - CSP, hereinafter PKCS #11);
- ***SCTOKEN.DLL*** - functions for working with a smart card through the interface PKCS #11;
- ***VTOKEN.DLL*** - functions for working with virtual slots and tokens through the interface PKCS #11;
- ***CRYPTOSP.DLL*** - a library of cryptographic procedures.

The above libraries are located in the WINDOWS \ SYSTEM32 folder. Also, the following modules are included in the scope of delivery of TCPI - CSP:

- ***GUI.DLL*** - interface for entering the password to the key, random number generation using the electronic roulette mechanism;
- ***CSP_INTEGRAL_TEST.EXE*** integral test for TCPI - CSP;
- ***PKCS11INI.EXE*** initialization of virtual slots and tokens for the PKCS #11 interface;
- ***CRYPTOTEST.EXE*** - tests of cryptographic algorithms;
- ***KEYMANAGER.EXE*** - test program for obtaining and viewing certificates;
- ***KM.DLL*** - dynamic library for supporting the work of the test program ***KEYMANAGER.EXE***;
- ***CSPCON.DLL*** is a dynamic library for supporting the import and export of private keys in the PFX format.

Note. The ***KEYMANAGER.EXE*** program and two dynamic libraries supporting it are intended only for testing the operation of CSP with certificates and private keys.

11. CSP INTERFACE

The TCPI - CSP interface consists of two dynamic libraries ***CSP.DLL, CSPFUNC.DLL***, as well as an auxiliary test program ***CSP_INTEGRAL_TEST.EXE***.

The cryptographic interface TCPI - CSP was created in accordance with the requirements of the Microsoft Cryptography Service Provider (CSP) standard. A description of the CSP standard (with a detailed description of all functions) can be found on the Microsoft website (http://msdn.microsoft.com/en-us/library/aa380245(v=VS.85).aspx). The TCPI - CSP

[17] PKCS #11 - is one of the standards of the Public-Key Cryptography Standards (PKCS) family. It defines a platform-independent software interface for accessing cryptographic devices (smartcards, tokens, cryptographic accelerators, key servers and other means of cryptographic information protection).

interface was developed for the implementation of cryptographic algorithms of the Republic of Uzbekistan [1], [2], [3], Russian encryption algorithms and electronic signatures [6, 7, 8] using PKCS #11 [5] interface.

When developing the software implementation of the cryptographic interface of TCPI - CSP, the source texts of programs from the world - famous and freely distributed software package OpenSSL were widely used.

Using TCPI - CSP can be done directly, by loading the ***CSPFUNC.DLL*** library using the LoadLibrary mechanism and obtaining addresses of cryptographic functions using the GetProcAddress[18] command, or via the CryptoAPI[19] interface.

Input and output data

The cryptographic interface of TCPI-CSP was created in accordance with the requirements of the Microsoft Cryptography Service Provider (CSP). A description of the CSP standard can be found on the Microsoft website. A description of the input and output data for the CSP standard can also be found on the Microsoft website at the links below.

CSP connection functions:

Function & Description

CPAcquireContext *– Associates a key container with a CSP pointer.*

CPGetProvParam *– Displays CSP parameters.*

CPReleaseContext *– Frees pointer received by* ***CPAcquireContext****.*

CPSetProvParam *– Sets specific CSP parameters.*

Key generation and CSP key exchange functions:

Function & Description

CPDeriveKey – Creates a key from a password.

CPDestroyKey – Removes a key from memory.

CPDuplicateKey – Creates a copy of the key.

CPExportKey – Export key.

CPGenKey *– Generates a random key.*

CPGenRandom *– Generates random numbers.*

CPGetKeyParam *– Gets key parameters.*

CPGetUserKey *– Gets a user key pointer.*

[18] GetProcAddress - The GetProcAddress function retrieves the address of the exported function or variable from the specified dynamic link library (DLL).

[19] CryptoAPI - CryptoAPI is an application programming interface that provides Windows developers with a standard set of functions for working with a cryptographic provider. Included in the Microsoft operating systems. Most CryptoAPI features are supported starting from Windows 2000. CryptoAPI supports asymmetric and symmetric keys, that is, allows you to encrypt and decrypt data, as well as work with electronic certificates. The set of supported cryptographic algorithms depends on the specific cryptographic provider.

CPImportKey *– Imports key.*

CPSetKeyParam *– Sets key parameters.*

Encryption and decryption functions:

Function & Description

CPDecrypt – Decrypts encrypted text with a key for encryption.

CPEncrypt – Encrypts plaintext with an encryption key.

Hashing and EDS functions:

Function & Description

CPCreateHash – Creates a hash pointer.

CPDestroyHash – Deletes a hash pointer.

CPDuplicateHash – Creates a copy of the hash function.

CPGetHashParam – Gets the properties of the hash function.

CPHashData – Hashes data

CPHashSessionKey – Hashes session key.

CPSetHashParam – Sets hash options.

CPSignHash – Sets signature hash function.

CPVerifySignature – Sets signature verification hash function.

12. CONCLUSION REMARKS

This work is devoted to a brief description of the CSP software, which is designed to create encryption keys, private and public keys of an electronic digital signature, creating and authenticating EDS, hashing, encrypting and simulating data using the algorithms described in the State Standards of Uzbekistan. It can be used in telecommunications networks, public information systems, government corporate information systems by embedding into applications that store, process and transmit information that does not contain information related to state secrets, as well as in the exchange of information and ensuring the legal significance of electronic documents [11].

CSP includes the following functional components: a dynamically loadable library that implements a biophysical sensor of random numbers; dynamic library that implements cryptographic algorithms in accordance with the State Standards of Uzbekistan; module supporting the work with external devices; installation module that provides the installation of CSP in the appropriate environment of operation (environment).

CSP provides the creation of private and public EDS keys and encryption keys; creation and confirmation of authenticity of EDS according to the algorithms described in [2, 7]; the formation of derived encryption keys used by data encryption algorithms described in [4, 5]; work with key information stored on external media; hashing of memory areas and other data according to the algorithms described in [3, 6]; encryption of memory areas and other data in accordance with the data encryption algorithms described in [4, 5].

CSP provides support for identifiers of algorithms and parameters for the implementation of compatibility with third-party cryptographic providers in terms of the ability to work with

public-key certificates issued by third-party registration centers, provided they use the cryptographic algorithms described in [2, 3, 4, 5, 6, and 7]. The cryptography service provider provides the ability to work with digital certificates of public keys, which are structured binary in ASN.1 format, conforming to ITU-T X.509 v.3 standard and IETF RFC 5280, RFC 3739 Recommendations. CSP provides work with external key carriers such as USB-flash, eToken Aladdin (eToken PRO 72K (JAVA)). As part of CSP, modules are provided that provide for calling cryptographic functions through the Microsoft CryptoAPI 2.0 interface when running under Microsoft operating systems.

In accordance with the functional purpose, CSP provides the generation of public and private EDS keys, hashing keys, functional keys and encryption keys for use, respectively, in the algorithms described in [2, 3, 4, 5, 6, and 7]. In the application where the CSP will be integrated, an appropriate system for the manufacture and distribution of keys will be provided, which will be based on the functions of generation of CSP keys. CSP allows the use of a multi-level key protection model using random and derivative keys of key protection. Protection of keys is carried out on the basis of cryptographic transformations in accordance with the PKCS #5 standard [9] using the State Standards of Uzbekistan [3], [4] or the interstate standard [5]. CSP provides storage of key information on a hard disk drive (HDD) and/or external key storage devices, such as USB-flash, eToken Aladdin (eToken PRO 72K (JAVA)). CSP provides work with key containers containing: signature keys, encryption keys and additional information necessary to ensure the cryptographic protection of keys and ensure control of their integrity. To protect key information from substitution and/or distortion during its storage on HDD and external key carriers, as well as during distribution, key information is supplied with a checksum. In order to ensure the safe use of the CSP installed on the PC, organizational measures are provided, as well as software and hardware methods and means of protecting information are used to ensure that secret keys stored in the PC's memory during operation of the CSP are kept secret, as well as the CSP service parameters stored on the hard drive. CSP contains a component that allows you to verify the operation of cryptographic algorithms implemented in it. Functioning is carried out on the basis of test examples. To ensure the safe use of an application with a built-in CSP, mechanisms are provided to control the integrity of the CSP libraries. In CSP, a biophysical random number sensor is used to generate random binary sequences that implement the mechanism for generating secret digital signature keys, encryption keys, initialization vectors using various algorithms.

REFERENCES

[1] O`z DSt 1105: 2009. State standard of Uzbekistan. Information technology CRYPTOGRAPHIC PROTECTION OF INFORMATION. Data encryption algorithm.
[2] O`z DSt 1106: 2009. State standard of Uzbekistan. Information technology CRYPTOGRAPHIC PROTECTION OF INFORMATION. A Hash function.
[3] O`z DSt 1092: 2009. State standard of Uzbekistan. Information technology CRYPTOGRAPHIC PROTECTION OF INFORMATION. Processes of formation and verification of electronic digital signature.
[4] PKCS #5 v2.0: Password-Based Cryptography Standard. RSA Laboratories. March 25, 1999. [Electronic resource]. - Access mode: https://tools.ietf.org/html/rfc2898.
[5] Expansion of PKCS #11 for the use of Russian cryptographic algorithms. Moscow, 2008.
[6] GOST 28147-89 - Information processing systems. Cryptographic protection. An algorithm of cryptographic transformation.
[7] GOST R 34.11-94 - Information technology. Cryptographic protection of information. A Hash function.

[8] GOST R 34.10-2001 - Information technology. Cryptographic protection of information. Processes of formation and verification of electronic digital signature.

[9] RFC 4357 Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms.

[10] Aloev R.D., Nurullaev M.M. (2019) Cryptography Service Provider - Data Encryption. In Proceedings of the 10 th International Multi-Conference on Complexity, Informatics and Cybernetics: 12-15 March 2019; Orlando, Florida, USA. Edited by Nagib Gallaos, Hsing-Wie Chu, Jeremy Horne, Suzanne K. Lunsford, Belkis Sánchez, Michael Savoie; pp 127-131.

[11] Aripov M.M., Alaev R.H. (2019) Research of the application of the new cryptographic algorithms: applying the cipher algorithm O'zDSt1105:2009 for MS office document encryption. In Proceedings of the 5th International Conference on Engineering and MIS (ICEMIS '19): 06-08 June 2019; Astana, Kazakhstan. ACM, New York, NY, USA. https://doi.org/10.1145/3330431.3330434.