# Applying Custom Algorithms in Windows Active Directory Certificate Services

Alaev Ruhillo
Faculty of Applied Mathematics and Intelligent Technologies
National University of Uzbekistan named after Mirzo Ulugbek, NUUz
Tashkent, Uzbekistan

***Abstract*—The article presents a solution to the problem of not recognizing the O'zDst 1092:2009 algorithm by the operating system and the problem of using digital certificates generated using the O'zDst 1092:2009 algorithm and O'zDst 1106:2009 algorithm. These algorithms were adopted in 2009. But these algorithms are still not recognized by the operating system. For other cryptographic algorithms used in Windows, cryptographic service providers have been developed that provide cryptographic operation functions to other software. These cryptographic service providers do not support the above algorithms. From here it becomes necessary to develop the cryptographic provider supporting the O'zDSt 1106:2009 hashing algorithm and the O'zDSt 1092:2009 signature algorithm. But to work with digital certificates, one cryptographic provider is not enough. Special extensions are also required to encode and decode digital certificate data. Therefore, the development of an extension for cryptographic providers is given. Also, for managing digital certificates and key lifecycle, a method of integrating cryptographic providers with Windows Active Directory Certificate Services is presented. Developed cryptographic providers are composed of 3 types of providers such as hash provider, signature provider, and key storage provider. The architecture of the key storage provider, a method for secure storage of cryptographic keys, as well as key access control are proposed. The description of the O'zDst 1092:2009 algorithm and the implementation of the functions of the Key storage provider interface are shown.**

***Keywords*—*O'zDSt 1106:2009 hashing algorithm; the O'zDSt 1092:2009 signature algorithm; active directory certificate services; digital certificate; key access control; key storage provider***

## I. Introduction

Electronic digital signature technology is widely used to ensure the integrity and identification of the owner of an electronic document. Presently in Uzbekistan, tools and methods that allow using O'zDst 1092:2009 signature algorithm do not provide document signing, signature validation and key management, through standard interfaces such as CryptoAPI, Cryptography Next Generation API [1] and PKCS [2], [3], [4], [5], [6]. This raises the problem of using the O'zDst 1092:2009 algorithm in information systems, such as not recognizing the O'zDst 1092:2009 algorithm by the operating system, working with digital certificates generated using the O'zDst 1092:2009 algorithm and O'zDst 1106:2009 algorithm. In Windows, cryptographic service providers are used to work with digital certificates, signing an electronic document, checking the digital signature. Starting with Windows Vista, Microsoft is offering a new Cryptography Next Generation API (CNG API) [1] to perform cryptography operations for applications. Microsoft provides several CNG providers that work with CNG API.

The list of CNG providers developed by Microsoft is shown in Table I.

TABLE I. The List of CNG Providers Developed by Microsoft

| CNG Provider | Description |
|---|---|
| Microsoft Primitive Provider | It supports the following functions:<br>hashing with SHA1, SHA256, SHA384, SHA512, MD2, MD4, MD5 algorithms;<br>signing and signature verification with RSA, DSA and ECDSA algorithms;<br>symmetric encryption and decryption with AES, DES, 3DES, DESX, RC2 and RC4 algorithms;<br>asymmetric encryption and decryption with RSA algorithm;<br>key exchange with DH and ECDH algorithms;<br>random number generation. |
| Microsoft Software Key Storage Provider | It performs operations on key pairs, especially persistent keys. It creates, exports, imports, deletes and stores key pairs. Key pairs generated by the Microsoft Primitive Provider are not persisted, so the Software Key Storage Provider is used for persistent keys. |
| Microsoft SSL Protocol Provider | It performs key management operations in SSL and TLS. It is used to establish a secure connection using SSL and TLS on Windows. |
| Microsoft Smart Card Key Storage Provider | It is used to store keys and digital certificates on smart cards. It creates, exports, imports, deletes, and stores key pairs as Software Key Storage Provider, but unlike it stores keys on smart cards. |
| Microsoft Key Protection Provider | It provides secure storage of keys. Also it allows you to protect content to a group in an Active Directory forest. |

Vendors such as "CryptoPro", "Infoteks", "Validata", "Signal-COM" and "Lissi-Soft" develop CNG providers that support both algorithms of international standards and Russian cryptographic algorithms.

CNG providers such as CryptoPro CSP, ViPNet CSP, Signal-COM CSP, Validata CSP, Lissi CSP, Tumar CSP and AVEST CSP support GOST R 34.11-94, GOST 34.11-2012, GOST R 34.10-2001, GOST 34.10-2012 and GOST 28147-89 algorithms. In addition AVEST CSP supports STB RB 1176.1-99, STB RB 1176.2-99, STB 34.101.18-2009, STB 34.101.31-2011, STB 34.101.45-2013, STB 34.101.47-2017 and STB 34.101.50-2019 algorithms.

CNG also offers a mechanism for implementing a new cryptographic algorithm in the operating system. For each algorithm, a separate cryptographic provider is developed, which is registered in the system for a certain class of algorithms. To manage cryptographic keys of digital signature algorithm, the tools, that supports this algorithm, are required. Only some digital signature algorithms support such tools. These tools can be hardware or software.Some implementations of hardware key management provide hardware-level key management ca-

pabilities; signing and signature verification is performed in software.

Key management tools allow you to automate the process of using keys in information systems, such as generating, storing, exporting, importing, destroying keys, encryption and decryption with a key, and key access control. Public Key Infrastructure (PKI) provides cryptographic key management [7]. PKI is a collection of services for managing keys and digital certificates of users, systems and networks.

PKI uses public key cryptography to identification of electronic exchange participants, control the integrity of information.

A Certificate Authority (CA) is main part of the PKI that issues a certificate to validate the rights of users or systems requesting. It creates the certificate and signs it using the CA private key. Another important part of PKI is the Certificate repository. Certificate repository is a store of valid certificates and certificate revocation lists (CRLs). Applications check the validity of the certificate using the CRL in the repository.

PKI performs the following main functions:

1) *Registration* is the process of collecting information about a user and verifying its authenticity.
2) *Certificate Issuance*. Once the CA has signed the certificate, it is issued to the applicant and/or sent to the certificate store. CA affixes a validity period to the certificates, thus requiring periodic renewal of the certificate.
3) *Certificate Revocation*. A certificate can become invalid before expiration for various reasons: the user has left the company, changed his name, or if his private key has been compromised. Under these circumstances, the CA will revoke the certificate by entering its serial number on the CRL.
4) *Key Recovery*. A PKI function that allows to recover data or key.
5) *Key and certificate Lifecycle Management* - maintenance of certificates in PKI, including renewal, recovery and archiving of keys.

All international PKI standards are based on the ITU-T X.509 standard [8], which defines the format of the digital certificate and CRL.

In CNG to manage keys, the key storage providers are used. Windows provides Active Directory Certificate Services (ADCS) for working with key store providers. ADCS is PKI system for Windows clients. ADCS by default works with signature algorithms such as RSA[9], Diffie Hellman (DH)[10], [11], Elliptic Curve Diffie Hellman (ECDH)[12] and Elliptic Curve Digital Signature Algorithm (ECDSA) [13].

Only a key storage provider is not sufficient to perform cryptographic operations. The CNG provides the following types of providers for this purpose: hash provider, cipher provider, asymmetric encryption provider, random number generator provider, secret agreement provider, signature provider, key derivation provider. Applications interact with CNG providers through CNG routers that provide CNG APIs [14]. The CNG API is divided into two groups:

1) *BCrypt API* – CNG Cryptographic Primitive Functions for cryptographic operations such as hashing, signing and signature verification, random number generation, encryption, asymmetric encryption, key derivation;
2) *NCrypt API* – CNG Key Storage Functions for working with cryptographic keys and CNG SSL Provider Functions.

Each type of algorithms has its own type of CNG router. Several CNG providers can be implemented for the algorithm.

The organizational structure of this article is as follows. Section 2 refers to related work. Section 3 provides a description of the O'zDSt 1092: 2009 signature algorithm, such as key pair generation, signing, and verification process. Section 4 provides the system architecture of the created ARH Primitive provider and ARH Key Storage provider. Section 5 presents the OIDs of the algorithms and the main parameters of the algorithms. Section 6 describes the implementations of the Key Storage provider interface functions. Section 7 describes the implementations of the Provider extension interface functions. The results will be presented and discussed in Section 8. Finally, Section 9 contains the conclusion.

## II. Related Work

A review of Microsoft's next-generation providers and the analysis of their supporting algorithms, types of providers were discussed in [15]. Windows default CNG providers do not support signature algorithm O'zDSt 1092:2009 [15]. The design and implementation of the key storage provider, which provides management keys' life cycle, were discussed in [16]. But the detailed description and integration with the Active Directory certification service has not been discussed. K. Lee and others [17] analyzed the possible vulnerabilities of the CNG library. The structures, functionality, and security issues of CNG have been discussed in [18], [19]. Cryptographic modules are not only developed as a cryptographic provider. The design and implementation of such elliptic curve cryptographic electronic signature systems were discussed in [20], [21], [22], [23]. But such cryptographic modules are not compatible with CNG, and therefore such solutions are not suitable for new custom algorithms. The development of a cryptographic provider for the hashing algorithm and the encryption algorithm was considered in the work [24]. It proposes a method of applying the symmetric algorithm O'zDSt 1105:2009, and the hashing algorithm O'zDSt 1106:2009 to ensure the confidentiality of electronic documents in MS Office. The authors of [25] discussed the development of a cryptographic provider that supports the CryptoAPI interface. They gave a detailed review of the cryptographic transformations of the encryption algorithm O'zDst 1005:2009, presented the architecture and modules of the CryptoAPI provider that they developed. But the CryptoAPI is already being replaced by a more advanced CNG API.

The above works analyze CNG libraries, the security of their key storage, provide a review of CNG API, discuss the development of the Key storage provider and the CryptoAPI provider. But the application of the new algorithm presented in this article in ADCS was not provided, and the problem of not recognizing digital certificates based on the O'zDSt 1092:2009 signature algorithm has not been resolved.

Therefore, we created CNG providers that support the above algorithms and provider extensions to address the issue of Windows digital certificates not being recognized. This article presents the architecture of these providers, describes the functions and modules, and proposes a solution to the problem of unrecognizing digital certificates.

## III. The First Signature Algorithm of the O'zDst 1092:2009 Standard

O'zDSt 1092:2009 is the State Standard of Uzbekistan. The standard includes two algorithms for creating and verifying an electronic digital signature. In this article, the first algorithm of the standard is discussed.

### A. Definition

The following definitions are defined in the algorithm:

$M$– message to be signed, represented in binary code, arbitrary finite length; $p$ – module, prime number, for software cryptographic module $p < 2^{1023}$;
$q$ – a prime number that is a factor (prime factor) of $p-1$, where $2^{254} < q < 2^{256}$;
$R$ – a natural number, parameter;
$(x, u)$ – a pair of integers – the private key of the signature algorithm;
$(y, z)$ – a pair of integers – the public key of the signature algorithm;
$(r, s)$ – a pair of integers – the signature value of $M$;
$H(M)$ – hash function that computes the hash value of $M$;
$md$ – mode of signing; for the mode with a session key $md = 1$, and for the mode without a session key $md = 0$.
$R_1$ – an integer number – the control key
$y_1$ – an integer number – the session key

### B. Special Operations

$X^{\backslash e}\,(mod\ p)$ - the operation of raising the base $x$ to the power $e$ modulo $p$ with the parameter $R$. For example, for $e = 41$.

$x^{\backslash 41}(mod\ p) => x^{\backslash 32+8+1}(mod\ p) =$
$(((((x^{\backslash 2})^{\backslash 2})^{\backslash 2})^{\backslash 2} * *(x^{\backslash 2})^{\backslash 2})^{\backslash 2} * *x(mod\ p)$.

where $x^{\backslash 2}(mod\ p) \equiv x(2 + xR)(mod\ p)$.

$X * *Y(mod\ p)$ – the operation of multiplication of two integer numbers modulo $p$ with parameter $R$. It is defined as follows:

$$X * *Y mod\ p \equiv X + (1 + XR)Y mod\ p \tag{1}$$

‘

$X^{\backslash -1}$ – the operation of the modular inverse of an integer $X$ modulo $p$ with parameter $R$. It is defined as follows:

$$X^{\backslash -1} \equiv -X\,(1 + XR)^{-1}\,(mod\ p) \tag{2}$$

### C. Key Generation

Algorithm 1 uses a one-way function in a group with a parameter, which is used in multiplication, exponentiation, and group inversion. Each user of Algorithm 1 has a private key and a public key:

1) $(x, u)$ - private key numbers, randomly or pseudo-randomly generated integers satisfying the condition $1 < x, u < q$;
2) $(y, z)$ - public key numbers calculated by the formula:

$$y \equiv g^{\backslash x}(mod\ p) \tag{3}$$

$$z \equiv g^{\backslash u}(mod\ p) \tag{4}$$

where $g$ is a private or public parameter, which is an integer, calculated according to the formula:

$$g \equiv h^{(p-1)/q}(mod\ p) \tag{5}$$

where: $h < p$ – is a natural number satisfying the condition $g^{\omega}\,(mod p) \equiv 0$, $\omega$ in the range of values $1, \ldots, q$ if and only if $\omega = q$

### D. Sign Process

Input parameters: $M, md, x, u, R_1$

Output: if $(md = 0)$ $\{r, s\}$, else $\{r, s, y_1\}$

Step 1: $m = H(M),\ c = x$

Step 2: $k = H(m + (1 + mR)c)$

Step 3: if $(k = 0)$ then $c = c + 2$ and go to Step 2.

Step 4: $T \equiv g^{\backslash -k}(mod\ p)$ with parameter $R$

Step 5: $r \equiv m + (1 + mR)T(mod\ p)$

Step 6: if $(r = 0)$ then $k = k + 1(mod\ p)$ and go to Step 4.

Step 7: $s_1 \equiv k - rx(mod\ q)$

Step 8: if $(s_1 = 0)$ then $k = k + 1(mod\ p)$ and go to Step 4.

Step 9: $s \equiv s_1 u^{-1}(mod\ q)$

Step 10: if $(md = 0)$ then return $\{r, s\}$ as signature values, else go to Step 11.

Step 11: $r_1 \equiv R_1 + (1 + RR_1)r(mod\ q)$

Step 12: if $(r_1 = 0)$ then $k = k + 1(mod\ p)$ and go to Step 4.

Step 13: $x_1 \equiv (k - suR_1)r_1^{-1}(mod\ q)$

Step 14: if $(x_1 = 0)$ then $k = k + 1(mod\ p)$ and go to Step 4.

Step 15: $y_1 \equiv gR_1^{\backslash x_1}(mod\ p)$ with parameter $RR_1$.

Step 16: return $\{r, s, y_1\}$ as signature values.

### E. Signature Verification Process

Input parameters: $M, md, \{y, z\}, r, s$; and $y1$ if $md = 1$

Step 1: $m = H(M)$

Step 2: if $L(s) \leq L(q)$ and $L(r) \leq L(p)$ then go to Step 3 else signature is not valid

Step 3: $z_0 \equiv z^{\backslash s}(mod\ p)$ with parameter $R$

Step 4: $r' \equiv r(mod\ q)$

Step 5: $y_2 \equiv y^{\backslash r'}(mod\ p)$ with parameter $R$

Step 6: $z_1 \equiv z_0 + (1 + z_0 R)y_2(mod\ p)$

Step 7: $y_3 \equiv z_1 + (1 + z_1 R)r(mod\ p)$

Step 8: if $(md = 0)$ and $(m = y_3)$ then signature is valid else if $(md = 0)$ and $(m = y_3)$ then go to Step 9, else if $(m \neq y_3)$ then signature is not valid;

Step 9: $g_3 \equiv z_1 R_1^{-1}(mod\ p)$

Step 10: $s_1 \equiv sR_1(mod\ q)$

Step 11: $r_1 \equiv R_1 + (1 + RR_1)r'(mod\ q)$

Step 12: $z_2 \equiv z + R_1^{-1}(mod\ p)$

Step 13: $y_4 \equiv y_1(mod\ p)$

Step 14: $z_3 \equiv z_2^{\backslash s_1}(mod\ p)$ with parameter $RR_1$

Step 15: $y_5 \equiv y_4^{\backslash r_1}(mod\ p)$ with parameter $RR_1$

Step 16: $g_4 \equiv z_3(1 + z_3 RR_1)y_5(mod\ p)$

Step 17: if $(g_3 = g_4)$ then signature is valid else signature is not valid

## IV. System Architecture

Applications for working with cryptographic keys use the key storage provider, for symmetric encryption they use the cipher provider, for hashing data they use the hash provider, for signing and verifying the signature they use the signature provider, etc. In turn, the key storage provider uses the signature provider to generate, export and import keys, sign data and verify the signature. To encrypt data with simmetric algorithms it uses the cipher provider, to compute a hash value of data, it uses the hash provider. Therefore, a key storage provider and a custom algorithm provider were developed to integrate with ADCS. The custom algorithm provider implements three interfaces: hash interface for algorithm O'zDst 1006:2009, signature interface for algorithm O'zDst 1092:2009 and cipher interface for algorithm O'zDst 1005:2009. The architecture of the developed system consists of four main parts (Fig. **??**): ARH Primitive Provider – the custom algorithm provider, ARH Key Storage Provider – the key storage provider, Provider extension – the extension module of provider, PKCS #11 - a module that implements the functions of the PKCS #11 interface and cryptographic algorithms.

## V. Main Parameters

The following constants are defined for the algorithms:

"ARH Primitive Provider" – the name of the custom algorithm provider

"ARH Key Storage Provider" – the name of the key storage provider

"O'zDSt 1092:2009 Alg1" – the name of the 1st algorithm of the O'zDst 1092:2009 standard

"O'zDSt 1092:2009 Alg2" – the name of the 2nd algorithm of the O'zDst 1092:2009 standard

"O'zDSt 1006:2009 Alg1" – the name of the 1st algorithm of the O'zDst 1006:2009 standard

"O'zDSt 1006:2009 Alg2" – the name of the 2nd algorithm of the O'zDst 1006:2009 standard

"O'zDSt 1005:2009 Alg1" – the name of the algorithm of the O'zDst 1005:2009 standard

"O'zDSt 1106:2009 Alg1/1092:2009 Alg1" – the name of the double algorithms "O'zDSt 1092:2009 Alg1" and "O'zDSt 1106:2009 Alg1"

"O'zDSt 1106:2009 Alg2/1092:2009 Alg2" – the name of the double algorithms "O'zDSt 1092:2009 Alg2" and "O'zDSt 1106:2009 Alg2"

1.2.860.3.15.1.1.1.1 – the OID of the 1st algorithm of the O'zDst 1092:2009 standard for sign

1.2.860.3.15.1.1.1.1.1 – the OID of test parameters of the first algorithm of the O'zDst 1092:2009 standard for sign

1.2.860.3.15.1.1.2.1 – the OID of the 2nd algorithm of the O'zDst 1092:2009 standard for sign

1.2.860.3.15.1.1.2.1.1 – the OID of test parameters of the second algorithm of the O'zDst 1092:2009 standard for sign

1.2.860.3.15.1.3.1 – the OID of the 1st algorithm of the O'zDst 1006:2009 standard

1.2.860.3.15.1.3.1.1 – the OID of test parameters of the first algorithm of the O'zDst 1006:2009

1.2.860.3.15.1.3.2 – the OID of the 2nd algorithm of the O'zDst 1006:2009 standard

1.2.860.3.15.1.3.2.1 – the OID of test parameters of the second algorithm of the O'zDst 1006:2009

1.2.860.3.15.1.1.1.2.2.1 the OID of the double algorithm "O'zDSt 1106:2009 Alg1/1092:2009 Alg1"

1.2.860.3.15.1.1.2.2.2.2 the OID of the double algorithm "O'zDSt 1106:2009 Alg2/1092:2009 Alg2"

The OIDs of algorithms and parameters are registered with the operating system.

## VI. Implementing Key Storage Provider Interface Functions

The following functions are implemented according to the requirement for the key storage provider interface:

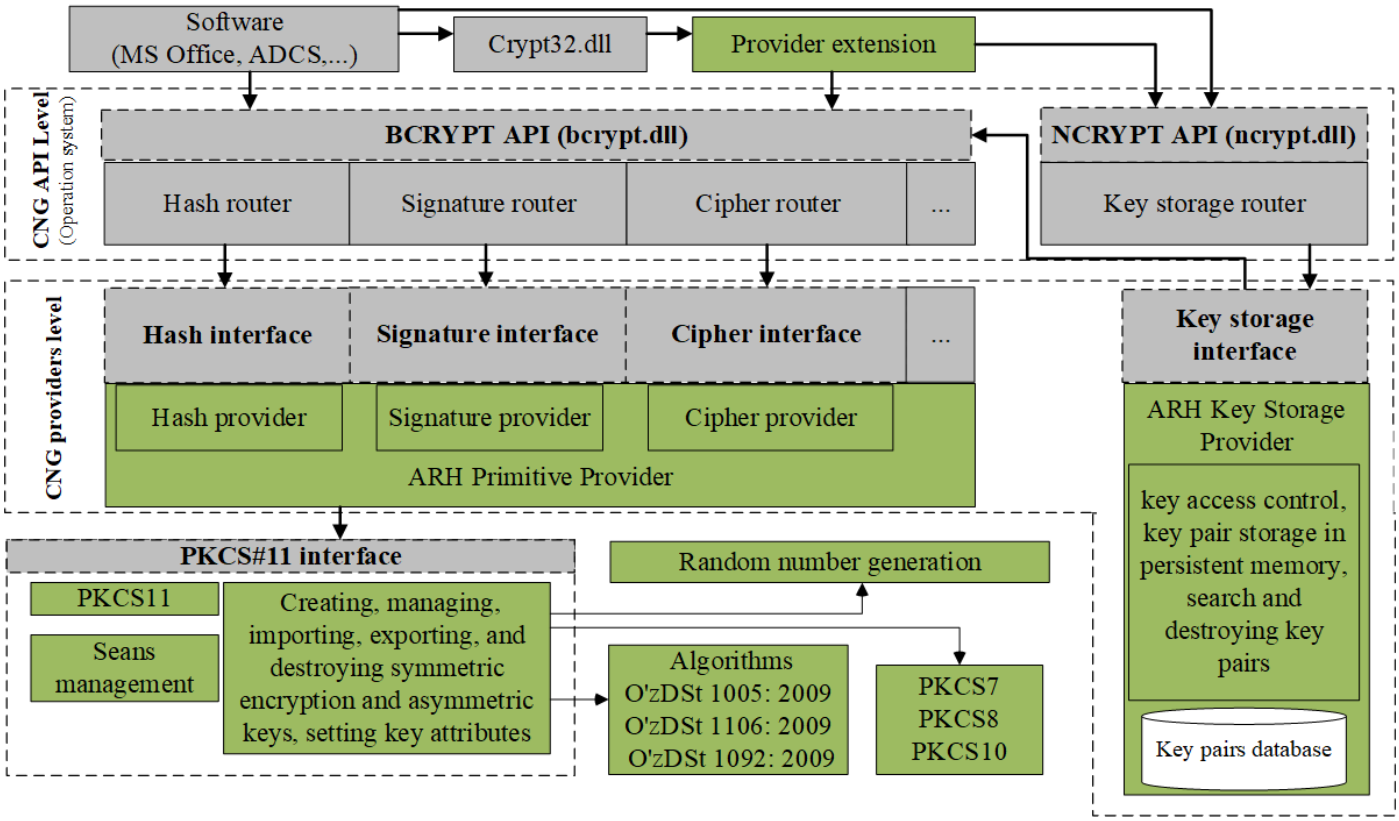


Fig. 1. The System Architecture.

- *GetKeyStorageInterface* function
- Key storage provider interface functions:
  - *OpenProvider*
  - *OpenKey*
  - *CreatePersistedKey*
  - *GetProviderProperty*
  - *GetKeyProperty*
  - *SetProviderProperty*
  - *SetKeyProperty*
  - *FinalizeKey*
  - *DeleteKey*
  - *FreeProvider*
  - *FreeKey*
  - *FreeBuffer*
  - *Encrypt*
  - *Decrypt*
  - *IsAlgSupported*
  - *EnumAlgorithms*
  - *EnumKeys*
  - *ImportKey*
  - *ExportKey*
  - *SignHash*
  - *VerifySignature*
  - *PromptUser*
  - *NotifyChangeKey*
  - *SecretAgreement*
  - *DeriveKey*
  - *FreeSecret*
  - *KeyDerivation*

The *GetKeyStorageInterface* function is used by the CNG router to get the address of the Key storage provider interface functions. The function takes the name of the key storage provider as an input parameter. An object of the *NCRYPT_KEY_STORAGE_FUNCTION_TABLE* structure is returned as an output parameter, which stores the addresses of the key storage provider interface functions. Later, the GNG router uses it to call interface functions of the Key storage provider.

The *OpenProvider* function is called by the CNG router when an application establishes a connection to the key storage provider. The function takes the name of the key storage provider as an input parameter and returns the handle of the provider. This handle serves as an identifier for the current connection. Also, this handle is used as an input parameter in many interface functions. The function initializes a provider object of type *ALPKSP_PROVIDER* and returns provider object addresses as handle of the provider.

*ALPKSP_PROVIDER* struct:

```
typedef struct _ ALPKSP_PROVIDER
{
        ALP_OBJECT_HEADER Header;
        //the size of the object
        // and magic number
        DWORD dwFlags;
        LPWSTR pszName;
        // the name of the provider
        BCRYPT_ALG_HANDLE hAlgorithm;
```

```
        //the handle of the
        //ARH Key Storage Provider
        LPWSTR pszContext;
        //context
} ALPKSP_PROVIDER
typedef struct _ALP_OBJECT_HEADER
{
        DWORD cbLength;
        // the size of the object
        DWORD dwMagic;
        // magic number
}
ALP_OBJECT_HEADER
```

The *CreatePersistedKey* function is called by the CNG router when generation of a key pair is required. It takes as input parameters the handle of the provider, the name of the algorithm, the name of the key, the type of key { *AT_KEYEXCHANGE* the key is a key exchange key, *AT_SIGNATURE* the key is a signature key, 0}, as well as a flag indicating the key of the current user or the local computer. If the key was created for the current user, then the user who created the key can use this key. In the case of a local computer, the key can be used by all users of the local computer. This method is often used in service applications. The function initializes a key object of type *ALPKSP_KEY*. The function will not generate key pair, it starts the key generation process of the key pair. Typically, after calling the *CreatePersistedKey* function, the *SetKeyProperty* function is used to specify the length of the key, and the value of other parameters. The key generation process ends with a call to the *FinalizeKey* function. To generate the key pair, the *CreatePersistedKey* function uses the ARH Primitive Provider through the signature interface and calls the *GenerateKeyPair* function.

Syntax of the *ALPKSP_KEY* struct:

```
typedef struct _ALPKSP_KEY
{
        ALP_OBJECT_HEADER Header;
        PALPKSP_PROVIDER hAlgoritm;
        DWORD dwKeyBitLen;
        BOOL isFinished;
        // whether the key is finalized
        BCRYPT_KEY_HANDLE hPublicKey;
        // the handle of public key
        BCRYPT_KEY_HANDLE hPrivateKey;
        // the handle of private key
        DWORD dwExportPolicy;
        // the export policy flag
        DWORD dwFlags;
        //{NCRYPT_MACHINE_KEY_FLAG,
        //NCRYPT_OVERWRITE_KEY_FLAG}
        LPWSTR pszKeyName;
        //the name of the key (key file)
        DWORD dwKeyUsagePolicy;
        //the key usage policy
        DWORD dwLegacyKeySpec;
        //the type of the key
        //{AT_KEYEXCHANGE, AT_SIGNATURE, 0}
        //encrypted private key blob
        PBYTE pbPrivateKey;
        DWORD cbPrivateKey;
        //hash value of pin
        PBYTE pbPinHash;
        DWORD cbPinHash;
        //handle to cryptography
        //providers needed to perform
        //operations with the key.
        BCRYPT_ALG_HANDLE hBCryptProvider;
        //security descriptor to be
        //set on the private key file.
        DWORD dwSecurityFlags;
        PBYTE pbSecurityDescr;
        DWORD cbSecurityDescr;
        NCRYPT_UI_POLICY_BLOB *pkeyUIPolicy;
        LIST_ENTRY PropertyList;
        //list of properties.
} ALPKSP_KEY;
```

The *FinalizeKey* function is called by the CNG router when an application needs to complete the key pair generation process. The function takes as input parameters the handle of the provider, the handle of the key. This function sequentially calls ARH Primitive Provider signature interface functions such as *FinalizeKeyPair* and *ExportKey*. The function encrypts the key blob and stores it on disk. If a pin code is specified, then the key is encrypted using the formula (6):

$$Ke = E(Kp, Kb) \tag{6}$$

$Kp$ – the encryption key, $Kb$ – the generated private key, $E$ – the O'zDst 1105:2009 encryption algorithm.

The encryption key $Kp$ calculated using the formula (7):

$$Kp = H(H(H(pin))) \tag{7}$$

$pin$ – the pin code value, $H$ – the O'zDst 1106:2009 hash function.

If the pin code is not specified, then the private key is encrypted using the *CryptProtectData* function.

The *"system"* and *"hidden"* attributes are set for the key file. If the key is applied to the local computer, full permission on the key file is assigned to the *System account* and the *Administrators group*, otherwise, to the current user account.

The *GetProviderProperty* function is used by the CNG router when an application needs to determine the value of the Key storage provider properties.

The *GetKeyProperty* function is used by the CNG router when an application needs to determine the value of the key properties. The function takes the handle of the key, property name as input parameters and returns the attribute value as an output parameter.

The *SetProviderProperty* function is used by the CNG router when an application needs to set the value of the Key storage provider properties.

The *SetKeyProperty* function is used by the CNG router when the value of the key attributes needs to be set. The function accepts the handle of the key, the attribute name, the new value of the attribute as input parameters.

The *OpenKey* function is called by the CNG router when an application opens an existing key. The function accepts as input parameters the handle of the provider, the key name, the key type, a flag indicating whether the key is for the current user or the local computer. The function initializes the key object of the *ALPKSP_KEY* type from the key file.

The *DeleteKey* function takes the handle of the key as an input parameter, deletes the key file from persistent storage, and also destroys the key object.

The *FreeProvider* function is called by the CNG router when an application closes the current connection to the key storage provider. The function takes the handle of the provider as an input parameter, frees the memory occupied by the provider object, which was created when calling the *OpenProvider* function. It closes the session with the pkcs11 module.

The *FreeKey* function takes the handle of the key as an input parameter and destroys the key object.

The *FreeBuffer* function takes a buffer address as an input parameter and frees memory.

The *Encrypt* function encrypts a block of data. The function uses the ARH Primitive Provider.

The *Decrypt* function decrypts the data block. The function uses the ARH Primitive Provider.

The *ImportKey* function imports the key that is exported by the *ExportKey* function. The function takes a key blob, a key blob type {*BCRYPT_PUBLIC_KEY_BLOB* – public key blob, *BCRYPT_PRIVATE_KEY_BLOB* – private key blob} as input parameters, and returns the handle of the key. If the key blob type is *BCRYPT_PRIVATE_KEY_BLOB*, then it saves the key to a file. The function initializes a key object of type *ALPKSP_KEY*.

The *ExportKey* function exports the key. The function takes as input parameters, the handle of the key, the key blob type, and returns the key blob through the *pbOutput* output parameter. The public key blob consists of the key identifier, key version, OID of the key, algorithm parameter values, and public key parameters. The private key blob consists of the key identifier, key version, OID of the key, algorithm parameter values, public/private key.

The *SignHash* function is used by the CNG router when an application needs to sign data. The function takes the handle of the key, a hash value as input parameters, and returns the generated signature. The function uses the ARH Primitive Provider.

The *VerifySignature* function is called by the CNG router when the signature needs to be verified. The function takes as input parameters the handle of the key, a hash value, the signature to be verified, and returns the verification result. The function uses the ARH Primitive Provider.

To integrate CNG providers with Windows Active Directory Certificate Services, they must be registered with the CNG router.

Therefore, the ARH Primitive Provider is registered to the hash, signature and cipher interface.

The ARH Key Storage Provider is registered to the key storage provider interface.

## VII. Implementation of Integration Functions

The Provider extension is developed to work with digital certificates. It implements the following callback functions:

The function *PFN_CRYPT_EXPORT_PUBLIC_KEY_INFO_EX2_FUNC* encodes and exports the public key blob. The function takes the handle of the key, an encoding type {*X509_ASN_ENCODING*, *PKCS_7_ASN_ENCODING*}, the public key as input parameters, and returns an object of type *CERT_PUBLIC_KEY_INFO*, which contains information about the public key. The function uses the ARH Key Storage Provider.

The function *PFN_IMPORT_PUBLIC_KEY_INFO_EX2_FUNC* decodes the public key algorithm identifier and imports the key. The function uses the ARH Primitive Provider.

The function *PFN_CRYPT_SIGN_AND_ENCODE_HASH_FUNC* signs and encodes the hash value. The function takes the handle of the key, encoding type, signature algorithm identifier, signature parameters, OID of the double algorithm, hash algorithm identifier, hash value as input parameters and returns the created signature. The function uses the ARH Primitive Provider.

The function *PFN_CRYPT_VERIFY_ENCODED_SIGNATURE_FUNC* decodes the signature and verifies the signature. The function takes as input parameters the type of encoding, the address of an object of type *CERT_PUBLIC_KEY_INFO*, which contains the public key, the OID of the signature algorithm, the identifier of the signature algorithm, signature parameters, the identifier of the hash algorithm, hash value, the signature to be verified, and returns the verification result. The function uses the ARH Primitive Provider.

These callback functions are called by the operating system to decode the algorithm OID fields in the certificate and to validate the certificate.

The provider extension is registered in the system using the *CryptRegisterOIDInfo* function to encode and decode the data of the digital certificate generated using the O'zDst 1092:2009 algorithm and O'zDst 1106:2009 hash algorithm.

## VIII. Result and Discussion

After registering CNG providers and provider extension, the Windows ADCS is installed and configured to work with the O'zDst 1092:2009 signature algorithm and O'zDst 1106:2009 hash algorithm.

Registered algorithm OIDs are located in the registry under the path HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptDllFindOIDInfo.

A Windows registry key is created for each algorithm OID. For example: [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\

CryptDllFindOIDInfo\1.2.860.3.15.1.1.1.2.2.1!4]
“Name” = “O’zDSt 1106:2009 Alg1/1092:2009 Alg1”
“Algid” = dword:ffffffff
“ExtraInfo” = hex:fe,ff,ff,ff
“Flags” = dword:00000001
“CNGAlgid” = “O’zDSt 1106:2009 Alg1”
“CNGExtraAlgid” = “O’zDSt 1092:2009 Alg1”

This registry key is created for the double algorithm(hash and signature algorithm) used in certificates. The key name includes the OID of the algorithm.

The registered provider extension functions are located in the registry under the following paths, respectively:

1) HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 1\CryptDllExportPublicKeyInfoEx2;
2) HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 1\CryptDllImportPublicKeyInfoEx2
3) HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 1\CryptDllSignAndEncodeHash
4) HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 1\CryptDllVerifyEncodedSignature

For each OID of the signature algorithm and dual algorithm, a key is created in the registry under the paths listed above. For example: [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 1\CryptDllVerifyEncodedSignature\1.2.860.3.15.1.1.1.2.2.1]
“Dll” = “C:\Program Files\ARHCrypto\ARHCNG\provext64.dll”
“FuncName” = “ARHVerifyEncodedSignature”.

“ARHVerifyEncodedSignature” is the name of the *PFN_CRYPT_VERIFY_ENCODED_SIGNATURE_FUNC* function implemented in the provider extension.

Registered providers are in the path HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Cryptography\Providers.

The comparison results by supported algorithms of Microsoft CNG providers with the created CNG providers are shown in Table II and Table III.

TABLE II. THE COMPARISON RESULTS OF MICROSOFT SOFTWARE KEY STORAGE PROVIDER AND ARH KEY STORAGE PROVIDER

| Functions | Microsoft Software Key Storage Provider | ARH Key Storage Provider |
|---|---|---|
| Generate, export/import, use and delete of key pairs of RSA, DSA and ECDSA algorithms | + | - |
| Generate, export/import, use and delete of key pairs of O’zDSt 1092:2009 algorithm | - | + |
| PIN based Authentication | - | + |
| Machine key type support | + | + |
| User key type support | + | + |
| Kernel mode(using from drivers) | + | - |
| User mode | + | + |

TABLE III. THE COMPARISON RESULTS OF MICROSOFT PRIMITIVE PROVIDER AND ARH PRIMITIVE PROVIDER

| Functions | Microsoft Primitive Provider | ARH Primitive Provider |
|---|---|---|
| Hashing with SHA1, SHA256, SHA384, SHA512, MD2, MD4, MD5 algorithms | + | - |
| signing and signature verification with RSA, DSA and ECDSA algorithms | + | - |
| symmetric encryption and decryption with AES, DES, 3DES, DESX, RC2 and RC4 algorithms | + | - |
| asymmetric encryption and decryption with RSA algorithm | + | - |
| key exchange with DH and ECDH algorithms | + | - |
| random number generation | + | + |
| signing and signature verification with O’zDSt 1092:2009 algorithm | - | + |
| hashing with O’zDSt 1106:2009 algorithm | - | + |
| symmetric encryption and decryption with O’zDSt 1105:2009 algorithm | - | + |
| Kernel mode(using from drivers) | + | - |
| User mode | + | + |

CNG providers, developed by Russian companies, work with their national algorithms, they also do not support O’zDSt 1092:2009, O’zDSt 1105:2009 and O’zDSt 1106:2009 algorithms.

A new certificate request has been created to test the system. Then a certificate is issued on request through ADCS.

The certificate content:

—BEGIN CERTIFICATE—
MIIDjDCCAtOgAwIBAgITLwAAAAc1JEX561oNaAAAAA
AABzAPBgsqhlwDDwEBAQICAQUAMBAxDjAMBgNVBA
MTBWFscENBMB4XDTIwMDgyMjExMTc1NloXDTIxMDg
yMjExMjc1NlowQTELMAkGA1UEBhMCVVoxDTALBgNV
BAoTBE5VVXoxIzAhBgkqhkiG9w0BCQEWFG1yLnJ1aGls
bG9AZ21haWwuY29tMIIBMDAlBgkqhlwDDwEBAQEwGA
YKKoZcAw8BAQEBAQYKKoZcAw8BAwEBAQOCAQUAB
IIBABQyfDPMWddMPpIIuRPC22IxhjlUw3ciPb2ugMQUsF8
ooshYXQyzkkL00BI8LQwmTEWcQJdGihZC0CCM+0KUYS
4A3aJbpjhnSWmkmE+v9gTfVUgUzdkA2uDcN4njUFJrSN7jz
/Fyo+IZutDze732QaIxC3ENwOkDtxNOsN/b//v/C6b7K/T0qsO
3LlJ1eH5VNF+rUdA9Ya9bRaLWzWE5UJGHz1NYJpSqld0R
8u6ft5Oo/lFPWWJH/S1qc1ku/tMw693Z8pVC4EzPzfJAyIsEO
V9O1KOkSKNJSI3wrE26pwZ8KDGkkWkC6WBwOxWMsu
IlbyHziNDuGdN6tyAUcBzbzbOjgfwwgfkwDgYDVR0PAQH/
BAQDAgSQMB0GA1UdDgQWBBREVPgsiLr5lss6Y5+uOR
w/An/jPzAfBgNVHSMEGDAWgBRBN7Hcr4RnJBtxH/f0i9d/
riFMTzA+BgNVHR8ENzA1MDOgMaAvhi1maWxlOi8vLy9
XSU4tN09FUkhLTFJDOUEvQ2VydEVucm9sbC9hbHBDQS
5jcmwwWQYIKwYBBQUHAQEETTBLMEkGCCsGAQUF
BzAChj1maWxlOi8vLy9XSU4tN09FUkhLTFJDOUEvQ2Vy
dEVucm9sbC9XSU4tN09FUkhLTFJDOUFfYWxwQ0EuY3J
0MAwGA1UdEwEB/wQCMAAwDwYLKoZcAw8BAQECAg
EFAAOBoQAhs9NcO8eECJIE8wmHlsNwL0HPB1aXpUolgk
NmN0mF8wpqFFRi90YREVsC3Kg/lTDy79yYNMZ8yRQ3Te
tE0Aj+w9n8T7THf9okYPDYE6xl4N+K4DQi+YulGyuPpzpLQ

Remr35snqXQj92KTJ8cgkkLjde5avD9V0eGVKKPNVPd4u5/
9wQPAoxn9sIiS1NuOwliwuyunVaVd6JgcyFC7Syq
——END CERTIFICATE——

By default Windows cannot verify the integrity of the certificate and cannot recognize the OID algorithms (Fig. **??**, Fig. **??**).

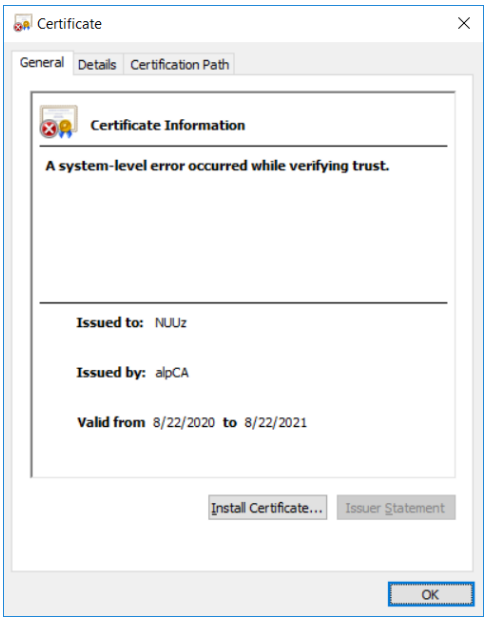


Fig. 2. Unrecognized Certificate.

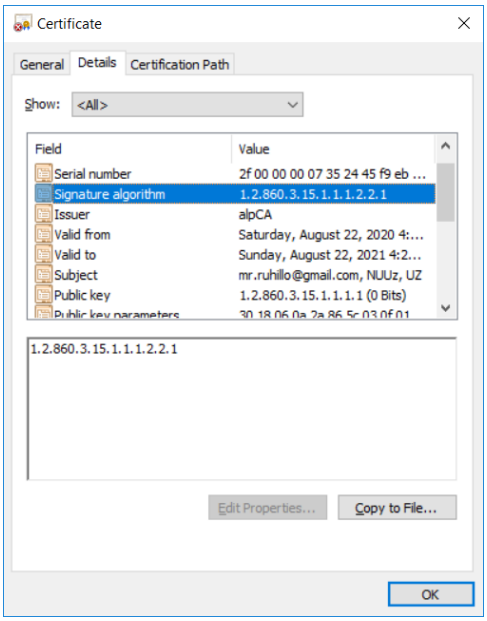


Fig. 3. Unrecognized Certificate Details.

This certificate is recognized by Windows that has both the ARH Primitive Provider and the ARH Key Storage Provider installed (Fig. **??**, Fig. **??**).

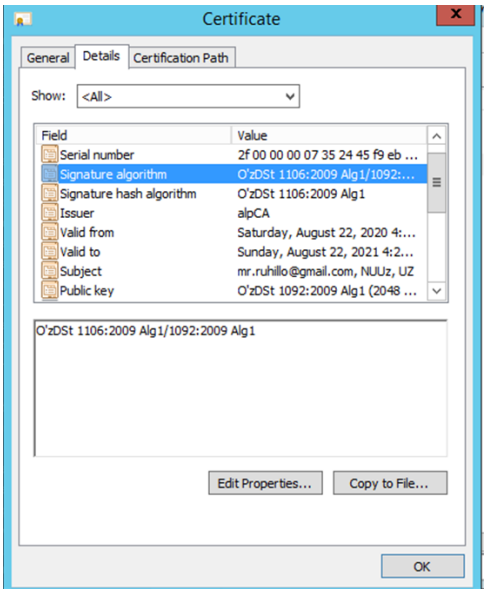


Fig. 4. Certificate Details.

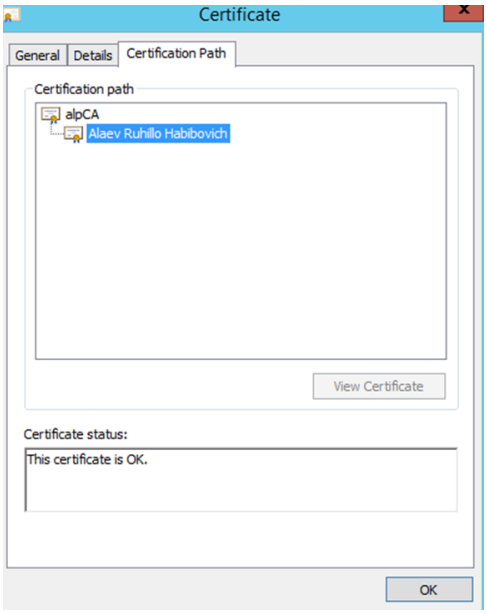


Fig. 5. Certification Path.

## IX. Conclusion

The architecture of the custom algorithm provider and key storage provider was provided. The description of the key storage provider interface functions was discussed. The implementation of the key storage provider interface functions was presented. The ARH Primitive Provider has been developed which implements the signature interface, the hash interface and the cipher interface, and also supports the O'zDst 1105:2009, O'zDst 1106:2009 and O'zDst 1092:2009 algorithms. The ARH Key Storage Provider has been developed, which implements the key storage provider interface. The ARH Key Storage Provider provides secure storage, use, export and import of national signature algorithm keys. The

ARH Key Storage Provider supports storage, export/import of signature keys in PKCS#7 and PKCS#8 formats, as well as generation of PKCS#10 requests for a digital certificate via the CertEnroll API. The description and implementation of the ADCS integration functions were provided.

The developed system solves the problem of not recognizing digital certificates generated based on the O'zDst 1106:2009 and O'zDst 1092:2009 algorithms. The CNG providers included in Windows by default do not address this issue.

The solution to the problem allows users to verify the integrity of the certificate generated based on the O'zDst 1106:2009 and O'zDst 1092:2009 algorithms. This, in turn, allows users to verify the integrity of data and documents signed with the private key of the certificate.

The methods proposed here can be used to apply other signature algorithms that are not supported by the Windows by default.

In addition, these CNG providers provide the ability to use these algorithms in existing information systems that work with other CNG providers. Because they already work with the CNG API to perform cryptographic operations. Sometimes this is achieved with just a few system settings, as a result, it is very easy to apply different algorithms in the system without additional costs and resources.

## ACKNOWLEDGMENT

I would like to acknowledge the reviewers for their valuable feedback.

## REFERENCES

[1] Singh, Abhishek. Identifying Malicious Code Through Reverse Engineering. 2009, 10.1007/978-0-387-89468-3.

[2] B. Kaliski. RFC2315: PKCS #7: Cryptographic Message Syntax Version 1.5. RFC Editor, 1998, USA.

[3] Kaliski, B. Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2. RFC, 5208, 2008.

[4] M. Nystrom and B. Kaliski. RFC2986: PKCS #10: Certification Request Syntax Specification Version 1.7. RFC Editor, 2000, USA.

[5] PKCS #11 Cryptographic Token Interface Base Specification Version 2.40, Committee Specification 01. OASIS Open (September 2014), http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/cs01/pkcs11-base-v2.40-cs01.html

[6] K. Moriarty, M. Nystrom, S. Parkinson, A. Rusch, and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1" RFC 7292, 2014.

[7] Johannes A. Buchmann, Evangelos Karatsiolis, and Alexander Wiesmaier. Introduction to Public Key Infrastructures. Springer Publishing Company, Incorporated. 2013.

[8] ITU-T Recommendation X.509 "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks ". 2000.

[9] Qi, N., Wei, W., Zhang, J., Wang, W., Zhao, J., Li, J., Hu, J. Analysis and research of the RSA algorithm. Information Technology Journal, vol.12(9), pp. 1818–1824, 2013, 10.3923/itj.2013.1818.1824.

[10] Diffie, W., Diffie, W., & Hellman, M. E. New Directions in Cryptography. IEEE Transactions on Information Theory, vol.22(6), pp. 644–654, 1976, 10.1109/TIT.1976.1055638.

[11] Maurer, U.M., Wolf, S. The Diffie–Hellman Protocol. Designs, Codes and Cryptography 19, pp. 147–171, 2000, 10.1023/A:1008302122286.

[12] Wei, W., Chen, J., Li, D., & Zhang, B. Research on the Bit Security of Elliptic Curve Diffie-Hellman. Journal of Electronics and Information Technology, vol.42(8), pp. 1820–1827, 2020, 10.11999/JEIT2_190845.

[13] M. Al-Zubaidie, Z. Zhang, J. Zhang, Efficient and secure ECDSA algorithm and its applications: A survey. International Journal of Communication Networks and Information Security. vol.11(1), pp. 7–35, 2019.

[14] Sean Turner and Russ Housley. Implementing Email and Security Tokens: Current Standards, Tools, and Practices. Wiley Publishing. 2008.

[15] Y. Ahmad. "A study on algorithms supported by CNG of Windows operating system" International Journal of Modern Engineering Research. vol.2(1) pp. 276–280, 2012.

[16] Z. Lina. "Design and implementation of KSP on the next generation cryptography API" Physics Procedia, International Conference on Medical Physics and Biomedical Engineering (ICMPBE2012). pp. 1640–1646, 2012.

[17] K. Lee, Y. Lee, J. Park, K. Yim, and I. You, "Security issues on the CNG cryptography library (Cryptography API: Next generation)" in 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. pp. 709–713, 2013.

[18] K. Lee, H. Lee, Y. Lee, and K. Yim, "Analysis on the key storage mechanism of the CNG library" in 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 499–502, 2016.

[19] K. Lee, I. You, and K. Yim, "Vulnerability analysis on the CNG crypto library" in 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 221–224, 2015.

[20] Khalique, A., Singh, K., & Sood, S. "Implementation of elliptic curve digital signature algorithm" International Journal of Computer Applications. vol.2(2) 2010.

[21] A. Abidi, B. Bouallegue, and F. Kahri, "Implementation of elliptic curve digital signature algorithm (ECDSA)" in 2014 Global Summit on Computer Information Technology (GSCIT), pp. 1–6, 2014.

[22] S. F. Temitope O.S. Olorunfemi, B.K. Alese and O. Fajuyigbe. "Implementation of elliptic curve digital signature algorithms" Journal of Software Engineering, vol.1(1), pp. 1–12, 2007. 10.3923/jse.2007.1.12.

[23] B. Chen, W. Wu, and Y. Zhang, "The design and implementation of digital signature system based on elliptic curve" in Proceedings of the 2012 International Conference on Cybernetics and Informatics, edited by S. Zhong (Springer New York, New York, NY, 2014) pp. 2041–2047.

[24] M.M. Aripov, R. H. Alaev. "Research of the application of the new cryptographic algorithms: Applying the cipher algorithm O'zDSt 1105:2009 for MS Office document encryption" in Proceedings of the 5th International Conference on Engineering and MIS, ICEMIS'19 (Association for Computing Machinery, New York, NY, USA, 2019). 10.1145/3330431.3330434

[25] M. Nurullaev and R. D. Aloev, "Software, algorithms and methods of data encryption based on national standards," IIUM Engineering Journal, vol.21(1), pp. 142–166, 2020. 10.31436/iiumej.v21i1.1179