ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 6, Issue 3, March 2019

Design and implementation the signature provider of the algorithmO'zDst1092:2009

Mersaid M. Aripov, Ruhillo H. Alaev

Professor, Department of Applied mathematics and Computer Analyses, National University of Uzbekistan, Tashkent, Uzbekistan
Research scholar, Department of Cyber security, National University of Uzbekistan, Tashkent, Uzbekistan

ABSTRACT:Electronic digital signature technology is widely used for ensuring the integrity and identification of the owner of an electronic document. Presently in Uzbekistan, tools and methods that allow using digital signature algorithm O'zDst1092:2009 do not provide document signing, signature validation and key management, through standard interfaces such as CryptoAPI, Cryptography Next Generation API and PKCS#11. This raises the problem of using the O'zDst1092:2009 algorithm by many information systems, such as working with digital certificates which was generated using the O'zDst1092:2009 algorithm. This article discusses the method of the O'zDst1092:2009 digital signature algorithm implementation in Windows. A model of the signature provider, a review of the mathematical functions of the algorithm O'zDst1092:2009, as well as a description and implementation of functions of the signature provider will be presented.

KEY WORDS: digital signature, Cryptography Next Generation API, signature provider, O'zDst1092:2009 signature algorithm, key pair generation.

I.INTRODUCTION

Signature providers are used for working with digital certificates, document signing and signature verification in windows. Starting with Windows Vista, Microsoft offers a new Cryptography Next Generation API (CNG API)[11], which provides performing cryptographic operations for applications. CNG API also offers a mechanism for implementing the new cryptographic algorithms into the Windows. For each algorithm, a CNG provider is created and registered. Registration makes the CNG provider available for use with applications. To implement the signature algorithm in Windows, the signature provider is created and added to the list of signature function providers.

This paper discusses applying signature algorithm through the design and implementation of the signature provider of the O'zDst1092:2009 algorithm. The O'zDst1092:2009 standard includes 2 digital signature algorithms. The first algorithm of the standard O'zDst1092:2009 was implemented in the signature provider. The architecture of the signature provider is shown in Fig. 1.

II.RELATED WORK

Research in this area has been published by many scientists from around the world. They were engaged in software and hardware implementations of digital signature algorithms such as RSA, DSA, ECDSA, GOST R 34.10-2012.In [1] the use of new cryptographic algorithms through CryptoAPI, which is already in the newly developed information systems are rarely used, was presented. A review of Microsoft's next-generation providers and the analysis of their supporting algorithms, types of providers were discussed in [2]. From [2] found out that CNG providers installed by default in Windows do not support signature algorithm O'zDSt1092:2009. The design and implementation of the key storage provider, which provides management keys' life cycle, was discussed in [3]. However, the design and implementing of the signature provider is not discussed. The analyze possible security vulnerabilities on the CNG library was provided in [6], analyze the key storage mechanism of the CNG library is discussed in [5]. The structures, features, and programming techniques of CNG API, security issues of CNG API are introduced in [4]. The implementation of elliptic curve-based digital signature algorithms was presented in [7-10].

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 6, Issue 3, March 2019

Therefore, in our opinion, given the above mentioned importance, there is a need for research and implement of the CNG provider which perform cryptographic operations with the new cryptographic algorithms.

III. THE SIGNATURE ALGORITHM O'ZDST1092:2009

The following constants were defined for algorithm:

1.2.860.3.15.1.1.1.1 – OID of the first algorithm of the standard O'zDst1092:2009
1.2.860.3.15.1.1.1.1.1 – OID of parameters of the first algorithm of the standard O'zDst1092:2009
"O'zDSt1092:2009 Alg1" – signature algorithm name
"ARH Primitive Provider" – signature provider name

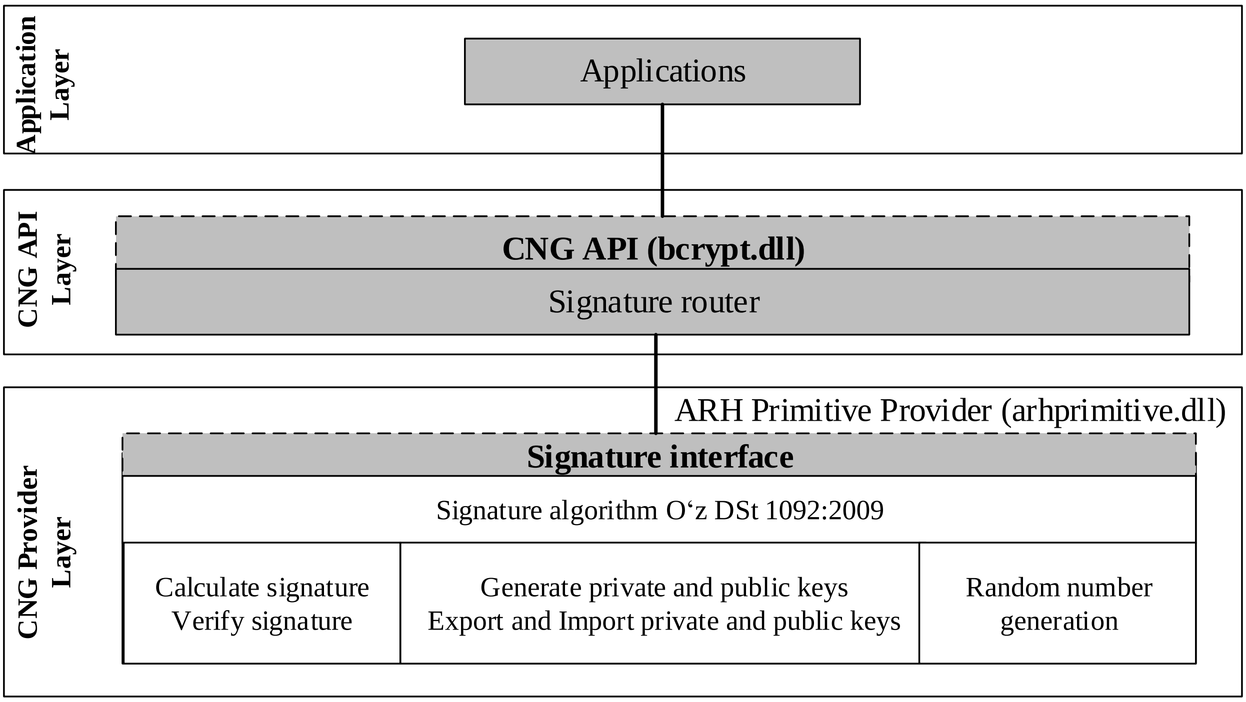

Fig. 1. The architecture of the signature provider

Mathematical functions

The following auxiliary functions were implemented for performing the operations specified in the algorithm:
The function *BN_xpR_mod_sqr(x,p,R)* calculates $X^{\backslash 2}(mod\ p)$with parameter R.

$$X^{\backslash 2}(modp) \equiv X(2+XR)\,(modp)$$

The function *BN_xypR_mod_mul(x,y,p,R)* - multiplication with parameter R modulo p, calculates $X \circledR Y\ (mod\ p)$ with parameter R.

$$X \circledR Y\ (mod\ p) \equiv X + (1+XR)Y\ (mod\ p)$$

The function *BN_xepR_mod_exp(x,e,p,R)* - the *e*-th power of *X* with parameter *R* modulo *p*, computes $X^{\backslash e}(modp)$ with parameter *R*. For example, for $e = 37$

$$X^{\backslash 37} => X^{\backslash 32+4+1}(mod\ p) \equiv \left(\left(\left(\left(\left(X^{\backslash 2}\right)^{\backslash 2}\right)^{\backslash 2}\right)^{\backslash 2}\right)^{\backslash 2} \circledR \left(X^{\backslash 2}\right)^{\backslash 2}\right) \circledR X\ (mod\ p)$$

Pseudocode of function:

bitsCount – bits count of *e*;
tmp1, tmp2, tmp3, i – integer numbers, at the beginning they are equal to zero
b[i] – i-bit of number *e*

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 6, Issue 3, March 2019

```
WHILE i<bitsCount
        if b[i] =1
        {
                r=BN_xypR_mod_mul(tmp1, tmp3, p,R)
                tmp1= r
        }
        tmp2=BN_xpR_mod_sqr(tmp3, p,R)
        tmp3=tmp2
        Increment i
ENDWHILE
    return r
```

Public and private key pair generation

a) Generating the pair 256-bit random numbers (*x*, *u*) – private key. $1 < x < q, 1 < u < q$

q = 0xA071C130A16485B29F52B17B952D1F590D758E62365494053BD0C1E71EE73011

b) Computing (*y*, *z*) – public key, *x* and *y* are both 1024-bit numbers:

$$y \equiv g^{\backslash x}(mod\ p)$$
$$z \equiv g^{\backslash u}(mod\ p)$$

p=0x1F84F3905B873C8B305375882F2EF26B346EFD236F20C76070AE1FB02EF773CD37DF3AA46463A97FADF E7672D53C6C53897C6D7A2C4255B5AA470AA3D0CD50FA5392D064BBFB6D7CEFB765B3266D264E3DF1811 C651A0E344957C154037048E5B24D9B9B67D684573EA08A242699C47A49DF55FD77B0DA4B449B37806CEDB F23

g=0x17B2927E70164CA06026C34C6A93DB2B6DFA0C90C981867DAE4F88E058D8DDD5E03FA615F1C667CC DB79641B0E4177499CFBE4393CB0EFC15994DD50B70A67DDC8CFA6DD2C9AD3CC844E90A9BE39679DC86 EFAAA21BF149F48916C4DBC3C8E7334B01C2636617E30A299BA8C4544B6C7DB895042CD7A04F7E8D6D202 89C83958

y= BN_xepR_mod_exp(g,x,p,R)

x= BN_xepR_mod_exp(g,u,p,R).

IV. DESIGN AND IMPLEMENTATION

Implementation of the digital signature algorithm O'zDSt1092:2009

The algorithm implementation has two main functions:

1) Digital signature calculation function: *SignAlg1*(IN *private key*(*x*,*u*), IN *hash value*, IN *μ-signature modeμ*∈{1, 0}, IN *control key* R_1, IN *values of the parameters* {*p*,*q*,*R*,*g*}, OUT *signature*, IN OUT *signature length*). The SignAlg1 function takes as input parameters the private key (*x*, *u*), a hash value, a signature mode, a control key, and calculates the signature based on these parameters. A block diagram of the signature calculation is shown in Fig. 2. For the mode with the session key $\mu = 1$, and for the mode without the session key $\mu = 0$.
2) Digital signature validation function: *VerifyAlg1*(IN *public key*(*y*,*z*), IN *hash value*, IN *μ-signature mode*, IN *control key*R_1, IN *values of the parameters* {*p*,*q*,*R*,*g*}, IN *signature*, IN *signature length*). The VerifyAlg1 function takes as input parameters the public key (*y*, *z*), the hash value, the signature mode, the control key, and verifies the signature based on these parameters. If the signature is valid, the *VerifyAlg1* function returns true else returns false. A block diagram of the signature validation is shown in Fig. 3.

Callback functions of the Signature Provider

According to the requirement of the signature provider interface, the following callback functions are implemented:

- Callback function*GetSignatureInterface*
- Signature interface callback functions
    - *OpenAlgorithmProvider;*
    - *GetProperty;*

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 6, Issue 3, March 2019

- *SetProperty;*
- *CloseAlgorithmProvider;*
- *GenerateKeyPair;*
- *FinalizeKeyPair;*
- *SignHash;*
- *VerifySignature;*
- *ImportKeyPair;*
- *ExportKey;*
- *DestroyKey.*

The callback function *GetSignatureInterface* is used by the CNG router in order to obtain the callback function's addresses of the signature interface. It takes the signature provider name and the algorithm name as input parameters. The callback function *GetSignatureInterface* returns an object of the *BCRYPT_SIGNATURE_FUNCTION_TABLE* structure as an output parameter, which stores the callback function's addresses of the signature interface.
Structure *BCRYPT_SIGNATURE_FUNCTION_TABLE*:

```
typedefstruct _BCRYPT_SIGNATURE _FUNCTION_TABLE {
        BCRYPT_INTERFACE_VERSION            Version;
        BCryptOpenAlgorithmProviderFn       OpenAlgorithmProvider;
        BCryptGetPropertyFn                 GetProperty;
        BCryptSetPropertyFn                 SetProperty;
        BCryptCloseAlgorithmProviderFn      CloseAlgorithmProvider;
        BCryptGenerateKeyPairFn             GenerateKeyPair;
        BCryptFinalizeKeyPairFn             FinalizeKeyPair;
        BCryptSignHashFn                    SignHash;
        BCryptVerifySignatureFn             VerifySignature;
        BCryptImportKeyPairFn               ImportKeyPair;
        BCryptExportKeyFn                   ExportKey;
        BCryptDestroyKeyFn                  DestroyKey;
} BCRYPT_SIGNATURE_FUNCTION_TABLE;
```

The *Version*, a member of structure contains the interface version value. In the current signature provider, it equals to *BCRYPT_SIGNATURE_INTERFACE_VERSION_1*. The remaining members of structure contain the addresses of the signature interface's callback functions.
The member's values of the object of the structure *BCRYPT_SIGNATURE_FUNCTION_TABLE*:

```
BCRYPT_SIGNATURE_FUNCTION_TABLE ALPSignatureFunctionTable = {
     BCRYPT_SIGNATURE_INTERFACE_VERSION_1,
     ALPOpenSignProvider,
     ALPGetSignProperty,
     ALPSetSignProperty,
     ALPCloseSignProvider,
     ALPGenerateSignKeyPair,
     ALPFinalizeSignKeyPair,
     ALPSignHash,
     ALPVerifySignature,
     ALPImportKeyPair,
     ALPExportKey,
     ALPDestroyKey
};
```

The *ALPOpenSignProvider* function is called by the CNG router when the application attempts to establish connection with the CNG provider. The general scheme of using CNG provider by applications is shown in Fig. 1.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 6, Issue 3, March 2019

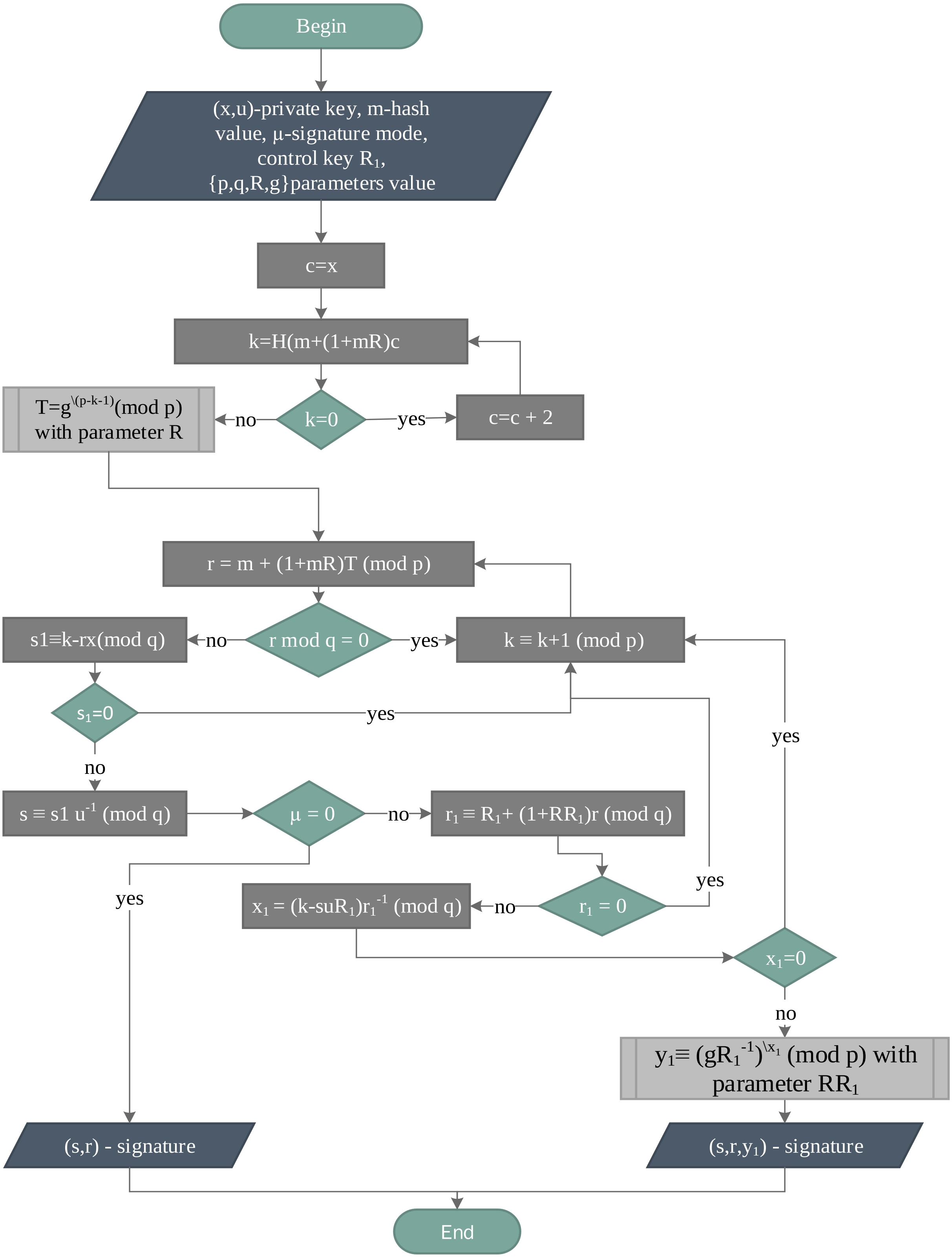

Fig.2.The block diagram of the signature calculation of the first algorithm of the O’zDSt1092:2009 standard

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 6, Issue 3, March 2019

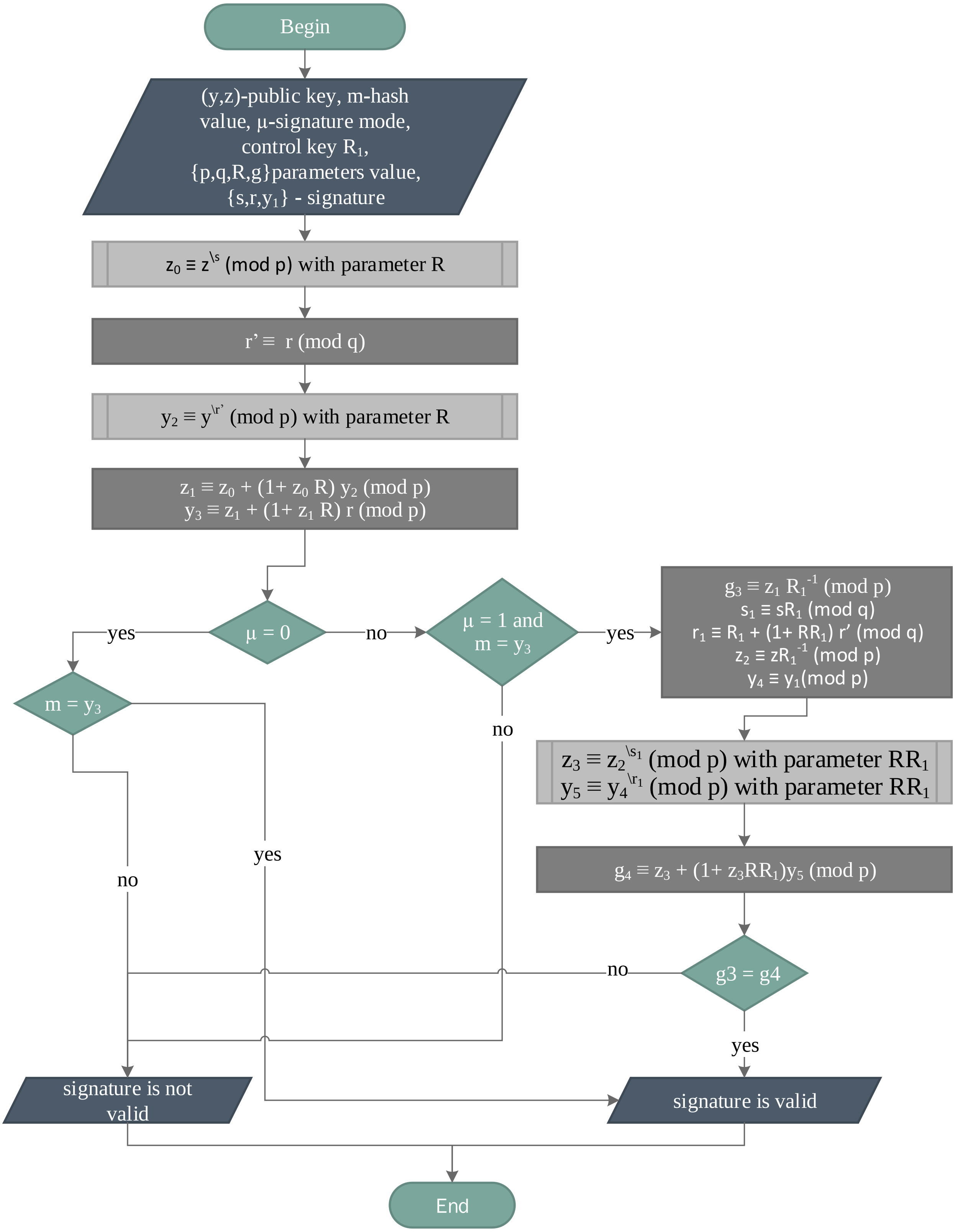

Fig.3. The block diagram of the signature validation of the first algorithm of the O'zDSt1092:2009 standard

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 6, Issue 3, March 2019

The *ALPOpenSignProvider* callback function takes the signature algorithm name as an input parameter, and returns the handle of the signature provider.This handle serves as an identifier of the current connection. Furthermore, it is also used as an input parameter in many functions of the signature interface. The callback function *ALPOpenSignProvider* initializes an object of the type *ALP_SIGN_ALGORITHM* and returns its address as a handle of the signature provider.
Structure *ALP_SIGN_ALGORITHM:*

```
    typedefstruct _ALP_SIGN_ALGORITHM{
        ALP_OBJECT_HEADER Header; //object length and magic number
        DWORD cbSignature; //signature length
        wchar_tszAlgorithmName[256]; // signature algorithm name
        DWORD dwKeyLen; //public key length in bits
        charparamOID[64]; // OID of the parameters
    } ALP_SIGN_ALGORITHM
typedefstruct _ALP_OBJECT_HEADER{
        DWORD       cbLength; // object length
        DWORD       dwMagic; // magic number
    }ALP_OBJECT_HEADER
```

The function *ALPGetSignProperty* is used by CNG router in order to retrieve the value of the property of a signature provider or a key. It accepts an object handle and the property name as input parameters, and returns the property value. An object handle can be a value of the type *BCRYPT_ALG_HANDLE*, which was obtained by calling the *ALPOpenSignProvider* callback function or a value of the type *BCRYPT_KEY_HANDLE*, which can be obtained by using the *ALPGenerateSignKeyPair*callback functionor the*ALPImportKeyPair* callback function. Usually, the callback function *ALPGetSignProperty* is called in order to obtain the signature algorithm name, signature length and the public key length.

The callback function *ALPSetSignProperty* is called by the CNG router in order to set the property value of the signature provider or the key. As input parameters, it assumes the followings: the object handle, the property name and a new value of the property.

The CNG router uses the callback function *ALPCloseSignProvider* when application closes the connection with the signature provider. It takes the handle of the signature provider as an input parameter. It frees the occupied memory by the signature provider object which was created by using the *ALPOpenSignProvider* callback function.

In order to generate a key pair, the callback function *ALPGenerateSignKeyPair* is called by the CNG router. As input parameters, it takes the handle of the signature provider and public key length, and returns the handle of the key. The callback function *ALPGenerateSignKeyPair* does not generate a key pair, it only begins the process of key pair generation. It initializes an object of type *ALP_SIGN_KEY* and returns the object address as a handle of the key. Usually, after calling the *ALPGenerateSignKeyPair* callback function, the *ALPSetSignProperty* callback function is called to set the key length, parameters p, q, R, g, μ, etc. The process of generating a key pair is completed by calling the *ALPFinalizeSignKeyPair* callback function.
Structure *ALP_SIGN_KEY:*

```
    typedefstruct _ALP_SIGN_KEY
    {ALP_OBJECT_HEADER Header;
    PALP_SIGN_ALGORITHM hAlgoritm;
    BYTE privateKeyX[16]; // x-private key parameter
    BYTE privateKeyU[16]; // u- private key parameter
    BYTE publicKeyY[128]; // y- public key parameter
    BYTE publicKeyZ[128]; // z- public key parameter
    DWORD dwKeyBitLen; // public key length
    BOOL isFinished; // whether the key is finalized
    BOOL mu; // signature mode
    LPCSTR pParamOid; // OID of the key
    BYTE *pbp; // p parameter
    DWORD *cbp; // p parameter length
```

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 6, Issue 3, March 2019

*BYTE *pbq; // q parameter*
*DWORD *cbq; // q parameter length*
*BYTE *pbR; // R parameter*
*DWORD *cbR; //R parameter length*
*BYTE *pbg; // g parameter*
*DWORD *cbg; // g parameter length,*
}ALP_SIGN_KEY

The callback function *ALPFinalizeSignKeyPair* is for complete key pair generation process. It takes the handle of the key as an input parameter, and calculates (x, u) - the two numbers of the private key, (y, z) - the two numbers of the public key. This callback function does not create a persistent key.

The *ALPSignHash* callback function is called by the CNG router in order to create a signature. It takes as input parameters the key handle, a hash value, and returns the created signature. In order to compute signature it uses the function *SignAlg1*.

The *ALPVerifySignature* callback function is called by the CNG router in order to verify the signature. It takes as input parameters the handle of the key, a hash value, a signature and returns the result of signature validation. In order to verify signature it uses the function *VerifyAlg1*.

The *ALPImportKeyPair* callback function imports a key that is exported by the *ALPExportKey* callback function. It takes as input parameters, a key *BLOB*, a type of *BLOB*, and returns the handle of the imported key. For this signature provider the type of *BLOB* can be one of the following values: *BCRYPT_PUBLIC_KEY_BLOB*, *BCRYPT_PRIVATE_KEY_BLOB*. The *ALPImportKeyPair* callback function initializes an object of type *ALP_SIGN_KEY*.

The *ALPExportKey* callback function exports the key. The *ALPExportKey* callback function as input parameters accepts the handle of the key, the type of BLOB, and returns the key BLOB. The public key BLOB consists of a key identifier, key version, OID of the key, (y, z) - values of the public key. The private key BLOB consists of the key identifier, key version, OID of the key, (y, z) - values of the public key, (u, x) - values of the private key.
The callback function *ALPDestroyKey* is used by CNG router to destroy a key. It takes the handle of the key as an input parameter.

V. REGISTRATION OF THE SIGNATURE PROVIDER

The function *BCryptRegisterProvider* is used for register the signature provider [3].
Syntax:

NTSTATUS WINAPI BCryptRegisterProvider(
_In_ LPCWSTR pszProvider,
_In_ ULONG dwFlags,
_In_ PCRYPT_PROVIDER_REG pReg)

The function *BCryptRegisterProvider* accepts the name of the algorithm provider via the *pszProvider* parameter, and the rest of the configuration data via the *pReg* parameter. The configuration data contains the algorithm name and the algorithm class.
After calling the *BCryptRegisterProvider* function, the *BCryptAddContextFunctionProvider* function is called in order to add the provider to the list of signature function providers.

VI. REGISTRATIONOID INFORMATION OF THE O'ZDST1092:2009 SIGNATURE ALGORITHM

As the algorithm identifier, not only the name of the algorithm is used, but also the OID of the algorithm is used. The name of the algorithm are not used in certificate signing request(CSR) and in digital certificate, the OID of the algorithm are used instead. That’s why the OID of the algorithm must be registered in Windows:

#define szOID_UZASYMM1_SIGN “1.2.860.3.15.1.1.1.1”

```
#define BCRYPT_UZ_SIGN_ALG1_2009 L"O'zDSt 1092:2009 Alg1"
// Register the O'zDSt 1092:2009 Alg1 Algorithm
    CRYPT_OID_INFO UZSIGN_ALG1OIDInfo;
    memset(&UZSIGN_ALG1OIDInfo, 0, sizeof(UZSIGN_ALG1OIDInfo));
    UZSIGN_ALG1OIDInfo.cbSize = sizeof(UZSIGN_ALG1OIDInfo);
    UZSIGN_ALG1OIDInfo.pszOID = szOID_UZASYMM1_SIGN; // OID for the sign Alg.
    UZSIGN_ALG1OIDInfo.pwszName = BCRYPT_UZ_SIGN_ALG1_2009;
    UZSIGN_ALG1OIDInfo.dwGroupId = CRYPT_PUBKEY_ALG_OID_GROUP_ID;
    UZSIGN_ALG1OIDInfo.Algid = CALG_OID_INFO_CNG_ONLY;
    UZSIGN_ALG1OIDInfo.pwszCNGAlgid = BCRYPT_UZ_SIGN_ALG1_2009;
    CryptRegisterOIDInfo(
            &UZSIGN_ALG1OIDInfo,
            0// dwFlags);
```

VII. RESULTS

The signature provider "ARH Primitive Provider", which implements the first algorithm of the standard O'zDSt1092:2009, was created. All functions of the signature provider interface, such as key generation, signature creation, signature verification, export and import keys was successfully tested on 32-bit and 64-bit Windows 8 and Windows 10 operating systems.

VIII. CONCLUSION AND FUTURE WORK

We proposed a model of a signature provider, the implementation of the mathematical functions of the digital signature algorithm O'zDst1092:2009. We presented the description and implementation of the signature interface functions and the registration of signature provider, which can be used for implement other signature algorithms and signature providers. The created signature provider is used with other CNG providers such as a hash provider and a key storage provider.

The Key Storage Providers are used in order to secure storage, export and import keys. The created Key storage provider, which provides storage, export/import keys in the PKCS#7 and PKCS#8 format, and the generation PKCS#10 certificate signing request via the CertEnroll API are testing.

REFERENCES

[1]Aloev R.D., Nurullaev M.M., Alaev R.H. "Working with the key information", International Journal of Information Research and Review Vol. 03, Issue, 11, November, 3217-3220, 2016.
[2]Y. Ahmad. "A study on algorithms supported by CNG of Windows Operating System". International Journal of Modern Engineering Research (IJMER). Vol.2, Issue.1, pp-276-280, Jan-Feb 2012.
[3]Z. Lina. "Design and Implementation of KSP on the Next Generation Cryptography API". International Conference on Medical Physics and Biomedical Engineering (ICMPBE), vol. 33, pp. 1640-1646, Sep. 2012, DOI= https://doi.org/10.1016/j.phpro.2012.05.264.
[4]K. Lee, Y. Lee, J. Park, I. You, K. Yim. "Security Issues on the CNG Cryptography Library (Cryptography API: Next Generation)". Proceedings of International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing(IMIS), pp. 709-713, Jul. 2013, DOI= https://doi.org/10.1109/IMIS.2013.128.
[5]K. Lee, H. Lee, Y. Lee and K. Yim, "Analysis on the Key Storage Mechanism of the CNG Library," 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 499-502, Fukuoka, 2016, DOI= https://doi.org/10.1109/IMIS.2016.103.
[6]K. Lee, I. Oh, S. Lee, K. Yim. Vulnerability Analysis on the CNG Crypto Library. The Journal of Korean Institute of Communications and Information Sciences. Vol.42 No.04, pp. 838-847, DOI= https://doi.org/10.1109/IMIS.2015.34.
[7] Akhalique, k.Singh, S.Sood, "Implementation of elliptic curve digital signature algorithm", International journal of computer applications,vo1.2, May 2010.
[8] Abidi, Abdessalem&Bouallegue, Belgacem&Kahri, Fatma. (2014). Implementation of elliptic curve digital signature algorithm (ECDSA). GSCIT 2014 - Global Summit on Computer and Information Technology. 1-6. https://doi.org/10.1109/GSCIT.2014.6970118.
[9] Temitope O.S. Olorunfemi, B.K. Alese, S.O. Falaki and O. Fajuyigbe, 2007. Implementation of Elliptic Curve Digital Signature Algorithms. Journal of Software Engineering, 1: 1-12, DOI: http://dx.doi.org/10.3923/jse.2007.1.12.
[10] Bin Chen, Wenliang Wu, and Yao Zhang, The Design and Implementation of Digital Signature System Based on Elliptic Curve", the 2012 International Conference on Cybernetics and Informatics, Vo1.l63, pp. 2041-2047, 2014, DOI: http://dx.doi.org/10.1007/978-1-4614-3872-4_260.
[11] Cryptography API: Next Generation. Retrieved November 11, 2018, from https://docs.microsoft.com/en-us/windows/desktop/seccng/cng-portal.