Journal of ELECTRICAL ENGINEERING, VOL 73(2022), NO3, 190–196

sciendo

PAPERS

Flawed implemented cryptographic algorithm in the Microsoft ecosystem

Stefan Pocarovsky, Martin Koppl, Milos Orgon[1]

With the continuous development in the electronic chip field, the requirements for the security of IT infrastructures are also increasing. The need for ever-increasing key lengths in cryptography to maintain security cannot grow indefinitely. One of the solutions in the field of cryptography for using shorter keys while maintaining security is cryptography based on the principle of elliptic curves. Asymmetric elliptic curve cryptosystems lies in solving the discrete logarithm problem on an elliptic curve. However, not only secure algorithm but also its correct implementation is important. In this paper, we discuss an incorrect implementation of the ECC algorithm in the crypt32.dll library (Microsoft Windows) and the possibilities of its misuse.

K e y w o r d s: ECC, cryptographic, security, crypt32. dll, ECCDSA

1 Introduction

The use of asymmetric cryptosystems versus symmetric ones is a leap forward. Basic uses of asymmetric cryptography include public-key encryption, *eg* , RSA [1], key exchange on an insecure channel [2], or digital signing. In addition to message secrecy, message authenticity is also needed in information systems security, where in addition to authenticity using symmetric cryptosystems [3], authenticity using asymmetric systems, *eg* , RSA-PSS [4], or a digital certificate [5], is much more efficient. The most widely used standard for a public key certificate is the X. 509 standard. This standard is compatible with various algorithms, such as the use of ECC [6]. ECC asymmetric cryptography is a suitable alternative to cryptographic systems that are based on the factorization problem due to its efficiency and lower computational complexity (due to smaller key lengths and the same security as, *eg* , RSA) [7]. The use of different software in implementing public key cryptography based on ECC in X. 509 certificates has properties that are often the target of cryptographic attacks. For example, one software dependency is the mechanism for validating and verifying the validity of certificates. In Windows operating systems, this feature is handled by CryptoAPI. We analyze the crypt32. dll library, which validates X. 509 public key certificates based on ECC. We address vulnerability CVE-2020-0601 in crypt32. dll in certain versions of the Windows operating system, which allows an attacker to forge an ECC-based X. 509 public key certificate without knowledge of the private key. Several detection methodologies are also proposed in [8].

Security is one of the most important characteristics that an information system must meet. The constant and ever more rapid development of technology has a major impact on the security of information exchange, the security of operating systems and applications and the security of computer networks. For example, cryptography is one of the most widely used methodologies for securing trusted communications. There are many cryptosystems that have stood the test of time against cryptographic attacks, but that is only if they have been used correctly. A very important parameter is the correct implementation of a particular crypto algorithm. If any parameter is neglected, selection of prime number, selection of correct elliptic curve (for ECC) for proper implementation of cryptographic system, the whole cryptosystem may be ineffective. Examples of these incorrect implementations are then system breaches, such as the poor implementation of crypt32.dll for ECC [8] or the past hack of the "Sony PlayStation 3" by the "FailOverFlow" group in 2010 [9].

2 Cryptography based on elliptic curves

For asymmetric cryptosystems based on elliptic curves, security rests are based on the mathematical problem of finding the discrete logarithm on the elliptic curve. Such cryptography is called "DL" cryptography. The question might arise why there is a talk about the problem of finding a discrete logarithm on an elliptic curve. For a group the name is not essential, and if instead "addition of points on an elliptic curve" we can use "multiplication of points on an elliptic curve", talking about the n- power of a chosen point. Then we can talk about the problem of the discrete logarithm, [10]. Such cryptography is mostly used in message authenticity checking (message signing).

[1] Slovak University of Technology in Bratislava, Faculty of Electrical Engineering and Information Technology, Bratislava, Slovakia, orgon@ktl.elf.stuba.sk, pocarovsky@gmail.com

https://doi.org/10.2478/jee-2022-0025, Print (till 2015) ISSN 1335-3632, On-line ISSN 1339-309X

For the same level of security (*eg* , versus RSA), it is sufficient to use a smaller group, *ie* , with EC-type cryptosystems. The requirements for computational power, storage, and message transmission can be thus reduced.

3 Elliptic curve

An elliptic curve is a plane, smooth and continuous curve defined by

$$y^2 = f(x), \tag{1}$$

where $f(x)$ - is a cubic polynomial in x with no multiple roots. Such an curve can be written in the Weierstrass (reduced) form

$$y^2 = x^3 + ax + b, \tag{2}$$

provided it is nonsingular *ie*

$$\Delta = -16(4a^3 + 27b^2) \neq 0. \tag{3}$$

The group of coordinates $\{x, y\}$ satisfying (2) are the individual points of a given elliptic curve. If R are real numbers, and $\{x, y\} \in R$, we have an elliptic curve over the real numbers. We define a zero point on the curve, as a point at infinity. For an elliptic curve to be usable in cryptography, it must be nonsingular. If the discriminant $\Delta = 0$, the curve is singular, having either a nodal or a spike singularity, as shown in Fig. 1.

Building a cryptosystem on a singular curve (*eg* a spike or node structure), the discrete logarithm of elliptic curves could be transformed into a classical discrete logarithm. This would reduce the level of security, [6].

Figure 2 shows examples of different shapes of elliptic curves.

EC cryptosystems are very similar to DL cryptosystems. Both use cyclic groups. While DL cryptosystems use a multiplicative group on the set of integers, EC cryptosystems use an additive group of points of an elliptic curve.

The security of cryptosystems using the elliptic curves is based on ECDLP (the discrete logarithm problem of elliptic curves). Compared to other asymmetric cryptosystems (*eg* , RSA) [7], the elliptic curve-based cryptography allows to use much shorter keys, with the same degree of security.

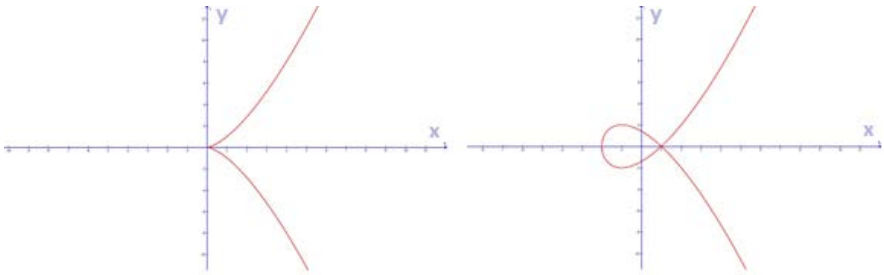

Fig. 1. Example of a singular curve of a spike structure (left) and a node structure (right)

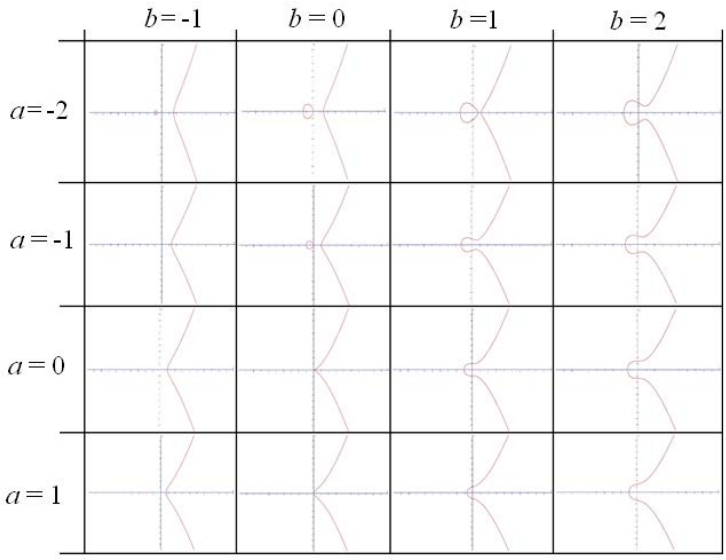

Fig. 2. Examples of different shapes of elliptic curves

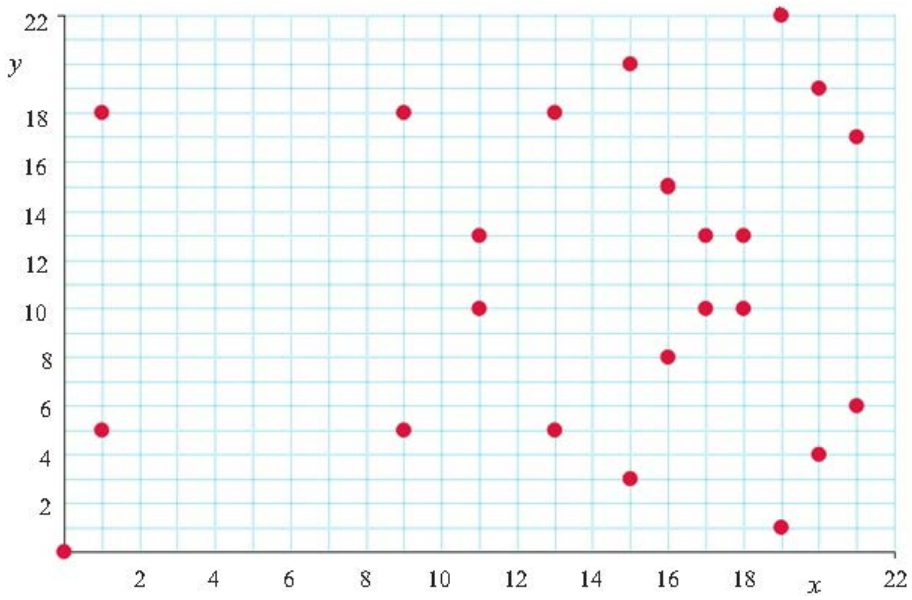

Fig. 3. Representation of an elliptic curve $y^2 = x^3 + ax + b$ withe $a = 1, b = 0, p = 23$

3.1 Elliptic curves over F_{p} and F_2^m

For EC, two types of finite fields are used, F_p is the elliptic curves over a field of primes and F_2^m, with $m \geq 1$, is a characteristic binary finite field. The elliptic curves over F_2^m have a finite number of points, hence the arithmetic can be performed efficiently, operating on bit strings. The arithmetic rules for m-bit strings of F_2^m can be defined by polynomial representation.

Every single point of an elliptic curve is given by the cartesian coordinates $P_i = \{x_i, y_i\}$.

Since in the cryptography based on elliptic curves an additive group is used, for each two points $P_i = \{x_i, y_i\}$ and $P_j = \{x_j, y_j\}$ on an elliptic curve over F_p, an addition operation is defined, that ma be written as

$$P_i, P_j \mapsto P_k. \tag{4}$$

The rules for this addition vary according to the actual situation, [11]. We describe the rules of summation of points on an elliptic curve for different input data in the equations below:

$$\begin{aligned}
&P_i, P_\infty \mapsto P_i,\\
&\text{adding point at infinity}\\
&\\
&\text{if: } (x_i = x_j) \wedge (y_i \neq y_j \vee y_i = y_j = 0)\\
&\text{then} \quad P_i, P_j \mapsto P_\infty,\\
&\\
&\text{else} \quad P_i, P_j \mapsto P_k, \quad \text{where:}\\
&y_k = \lambda(x_i - x_k) - y_i,\\
&\text{if: } (x_i \neq x_j) \text{ then:} \quad \lambda = \frac{y_j - y_i}{x_j - x_i}\\
&x_k = \lambda^2 - x_i - x_j,\\
&\\
&\text{if: } (x_i = x_j) \wedge (y_i \neq 0 \wedge y_j \neq_0) \text{ then:}\\
&\lambda = \frac{3x_i^2 + a}{2y_i}, \quad x_k = \lambda^2 - 2x_i.
\end{aligned} \tag{5}$$

In cryptography, the equation

$$y_i^2 \mod p = (x_i^3 + ax_i + b) \mod p, \tag{6}$$

may be used, where p as an integer divider and a, b are real numbers.

Example of an elliptic curve over F_{p} is in Fig. 3, providing $(4a^3 + 27b^2) \neq 0$. The solution of (6) with parameters $a = 1, b = 0$ and $p = 23$ corresponds to 24 points. For simplicity, we have chosen a small prime number, while for ECC cryptography, it is recommended to use elliptic curves standardized by BRAINPOOL or NIST, such as NIST P-256, NIST P-384, NIST P-521, or brainpool P256t1, brainpool P384t1, brainpool P512t1. All these elliptic curves (and others that can be considered secure according to the standards) are included in the operating systems in use that support ECC-based cryptography.

3.2 ECDSA

Elliptic curve digital signature algorithm (ECDSA) is a digital signature cryptosystem that is based on elliptic curves. It relies on the mathematics of cyclic hump over finite fields and the discrete logarithm problem on elliptic

```
Curve Name          Curve OID                  Public Key Length   CurveType           EccCurveFlags
-------------------------------------------------------------------------------------------------
curve25519                                     255                 29                  0xa
nistP256            1.2.840.10045.3.1.7        256                 23                  0x7
nistP384            1.3.132.0.34               384                 24                  0x7
brainpoolP256r1     1.3.36.3.3.2.8.1.1.7       256                 26                  0x7
brainpoolP384r1     1.3.36.3.3.2.8.1.1.11      384                 27                  0x7
brainpoolP512r1     1.3.36.3.3.2.8.1.1.13      512                 28                  0x7
nistP192            1.2.840.10045.3.1.1        192                 19                  0x7
nistP224            1.3.132.0.33               224                 21                  0x7
nistP521            1.3.132.0.35               521                 25                  0x7
secP160k1           1.3.132.0.9                160                 15                  0x7
secP160r1           1.3.132.0.8                160                 16                  0x7
secP160r2           1.3.132.0.30               160                 17                  0x7
secP192k1           1.3.132.0.31               192                 18                  0x7
secP192r1           1.2.840.10045.3.1.1        192                 19                  0x7
secP224k1           1.3.132.0.32               224                 20                  0x7
secP224r1           1.3.132.0.33               224                 21                  0x7
secP256k1           1.3.132.0.10               256                 22                  0x7
secP256r1           1.2.840.10045.3.1.7        256                 23                  0x7
secP384r1           1.3.132.0.34               384                 24                  0x7
secP521r1           1.3.132.0.35               521                 25                  0x7
```

Fig. 4. Extract elliptic curves from Win 10 Enterprise (ver. 20H2)

```
ECC Curve Parameters
--------------------
    Version: 1

    FieldID:
        FieldID Type OID: 1.2.840.10045.1.1    (Prime Field)

        Prime P:
        0000  ff ff ff ff 00 00 00 01  00 00 00 00 00 00 00 00
        0010  00 00 00 00 ff ff ff ff  ff ff ff ff ff ff ff ff

    Curve:
        A:
        0000  ff ff ff ff 00 00 00 01  00 00 00 00 00 00 00 00
        0010  00 00 00 00 ff ff ff ff  ff ff ff ff ff ff ff fc

        B:
        0000  5a c6 35 d8 aa 3a 93 e7  b3 eb bd 55 76 98 86 bc
        0010  65 1d 06 b0 cc 53 b0 f6  3b ce 3c 3e 27 d2 60 4b

    Base:
        0000  04 6b 17 d1 f2 e1 2c 42  47 f8 bc e6 e5 63 a4 40
        0010  f2 77 03 7d 81 2d eb 33  a0 f4 a1 39 45 d8 98 c2
        0020  96 4f e3 42 e2 fe 1a 7f  9b 8e e7 eb 4a 7c 0f 9e
        0030  16 2b ce 33 57 6b 31 5e  ce cb b6 40 68 37 bf 51
        0040  f5

    Order:
        0000  ff ff ff ff 00 00 00 00  ff ff ff ff ff ff ff ff
        0010  bc e6 fa ad a7 17 9e 84  f3 b9 ca c2 fc 63 25 51

    Cofactor:
        0000  01
```

Fig. 5. Parameters of elliptic curve nistP256 from Win 10 Enterprise (ver. 20H2)

curves. The signing algorithm depends on the multiplication of points on the elliptic curve. Keys and signatures are shorter than in RSA (with the same level of security). This cryptosystem is of the authentication type. It is used to determine the authenticity of a message.

3.3 CVE-2020-0601

On 14 January 2020, the NSA (National Security Agency) published a document [12] identifying a critical vulnerability in Windows systems in the crypt32.dll library. The vulnerability could be exploited in X.509 certificate authentication, where an attacker could forge X.509 certificates in a way that Windows would validate them as legitimately signed by a trusted entity [13]. The implications of this flawed algorithm implementation were significant for system security in at least two respects:

- Accesses to HTTPS-secured web pages would appear trusted because of the forged X.509 certificate. Currently, all web browsers authenticate TLS certificates against a trusted entity repository. The operating system would then verify the forged certificate as legiti-

mately signed from trusted entities. This would cause the spoofed (untrusted) website to appear trusted and secure during TLS certificate verification.

- Many of the security measures of current operating systems are based on the authenticity of the code being run. An attacker could create a certificate (exploiting the CVE-2020-0601 vulnerability) to sign executable code, the operating system would evaluate the certificate as trustworthy. Then, after such authentication, the necessary malicious code could be installed without problems.

This flawed implementation affects the following system versions:

- windows_10: 1607:*:*:*:*:*:*:*
- windows_10: 1709:*:*:*:*:*:*:*
- windows_10: 1803: *: *: *: *: *: *: *
- windows_10: 1809:*:*:*:*:*:*:*
- windows_10: 1903:*:*:*:*:*:*:*
- windows_10: 1909:*:*:*:*:*:*:*
- windows_server_2016:-:*:*:*:*:*:*:*:*
- windows_server_2016: 1803:*:*:*:*:*:*:*:*
- windows_server_2016: 1903:*:*:*:*:*:*:*:*
- windows_server_2016: 1909:*:*:*:*:*:*:*:*
- windows_server_2019:-:*:*:*:*:*:*:*

The vulnerability is implemented in the Microsoft Windows system in the CryptoApi module in the crypt32.dll library [13]. The ChainGetSubjectStatus() method in the crypt32.dll library is the basic component that validates the current certificate. However, no further comparison of parameters between the elliptic curve certificate and the CA certificate parameters in the system store was performed. This caused the elliptic curve certificate to have different parameters from the corresponding CA certificate and the system to evaluate it as trusted. With the ability to modify the elliptic curve parameters, an attacker can create a certificate of a trusted CA without knowing the private key kprivate. The Windows operating system implements elliptic curves and their parameters that have been evaluated by organizations such as NIST or Brainpool as secure for use in cryptography. Figure 4 shows a direct output from Windows 10 (using certutil) of some such curves.

In Figure 5 all the parameters of the curve are defined, even the base point "P". The basic principle of finding the discrete logarithm on an elliptic curve is to add the points on the elliptic curve

$$P_i, P_j \mapsto P_k, \quad \text{with: } P_i = P_j = P, \tag{7}$$

where, P is the agreed starting point. We thus consider n-additions of the same point P as

$$\underbrace{P, P \mapsto P \ldots P, P \mapsto P}_{n} \mapsto P_k \quad . \tag{8}$$

Given P and $n = k_{\text{pri}}$, where n is the number of additions of the base point of P, the private key k_{pub} can be formally computed as

$$k_{\text{pub}} = P\, k_{\text{pri}} \Longrightarrow k_{\text{pri}} = \frac{k_{\text{pub}}}{P} \quad . \tag{9}$$

For the computation of k_{pri} from (9), there is currently not known algorithm to determine it in a reasonable time (hours, days, weeks). However, if the algorithm allows for an arbitrary base point P, the difficulty of determining the value of k_{pri} by (9) is substantially decreased. In CVE-2020-0601, this flaw has been implemented where the ChainGetSubjectStatus() method in the crypt32.dll library does not verify the agreed P. Consequently, an attacker can experiment with a suitable starting point P to satisfy the k_{pri} requirement. He can create a certificate of a trusted CA, while using an arbitrary k_{pri} value and a suitably chosen starting point P satisfying the k_{pub} key value of (an already trusted CA) certificate. For this purpose, it is enough to use $P = k_{\text{pub}}/k_{\text{pri}}$.

As a result, it is possible to spoof a fake CA certificate, where an attacker chooses an arbitrary k_{pri} key value, computes a valid base point P in accordance with the k_{pub} key, the crypt32.dll library evaluates it as a trusted CA certificate to complete the certificate signing request.

Immediately after the release of the patch for the crypt32.dll library, in [14]was published possible exploit code for the vulnerability [14]. Author described the code and procedure for implementing the signing of an executable in the Microsoft ecosystem (without patching the CVE-0601-2020 bug) and establishing an HTTPS connection with the certificate thus spoofed. Due to a flawed implementation of the crypt32.dll library that does not verify the chosen base point P on the elliptic curve.

Choosing $k_{\text{pri}} = 1$ and using (9) may do it trivial to spoof the CA's public key certificate without knowing the private key k_{pri}. In a result, the certificate being verified as trusted by the crypt32.dll library.

3.4 Bug fix of crypt32.dll

After the NSA disclosed the flaw, MICROSOFT released a patch that fixes the flaw. This patch should modify the crypt32.dll library, which modifies or adds authentication of CA certificates. That library should add authentication that not only checks the value of the k_{pub} key, but also compares the value of the base point P. The following section worked with CryptoAPI, specifically the CRYPT32.dll libraries. The first library was downloaded before the patch update, the second was downloaded after the patch update, where the base point P verification bug was supposedly fixed. The third library, CRYP32.dll, was downloaded from Win 10 Pro 20H2. The goal was to compare all libraries, then evaluate the libraries in question and determine how the cryp32.dll patch was implemented to evaluate the base point P. The CRYPT32.dll library that was downloaded from Win 10 Pro 20H2 was added to check if the library in question was evolving over time.

```
10.04.2022  16:36    <DIR>          .
10.04.2022  16:36    <DIR>          ..
29.01.2022  10:50         1 383 152 crypt32-20H2.dll
01.03.2020  20:00         1 330 952 crypt32-afterPatch.dll
17.12.2019  11:49         1 327 064 crypt32beforePatch.dll
               3 File(s)      4 041 168 bytes
               2 Dir(s)  433 249 005 568 bytes free
```

Fig. 6. Comparison of crypt32.dll library file sizes before and after patch implementation and with Win 10 Pro 20H2

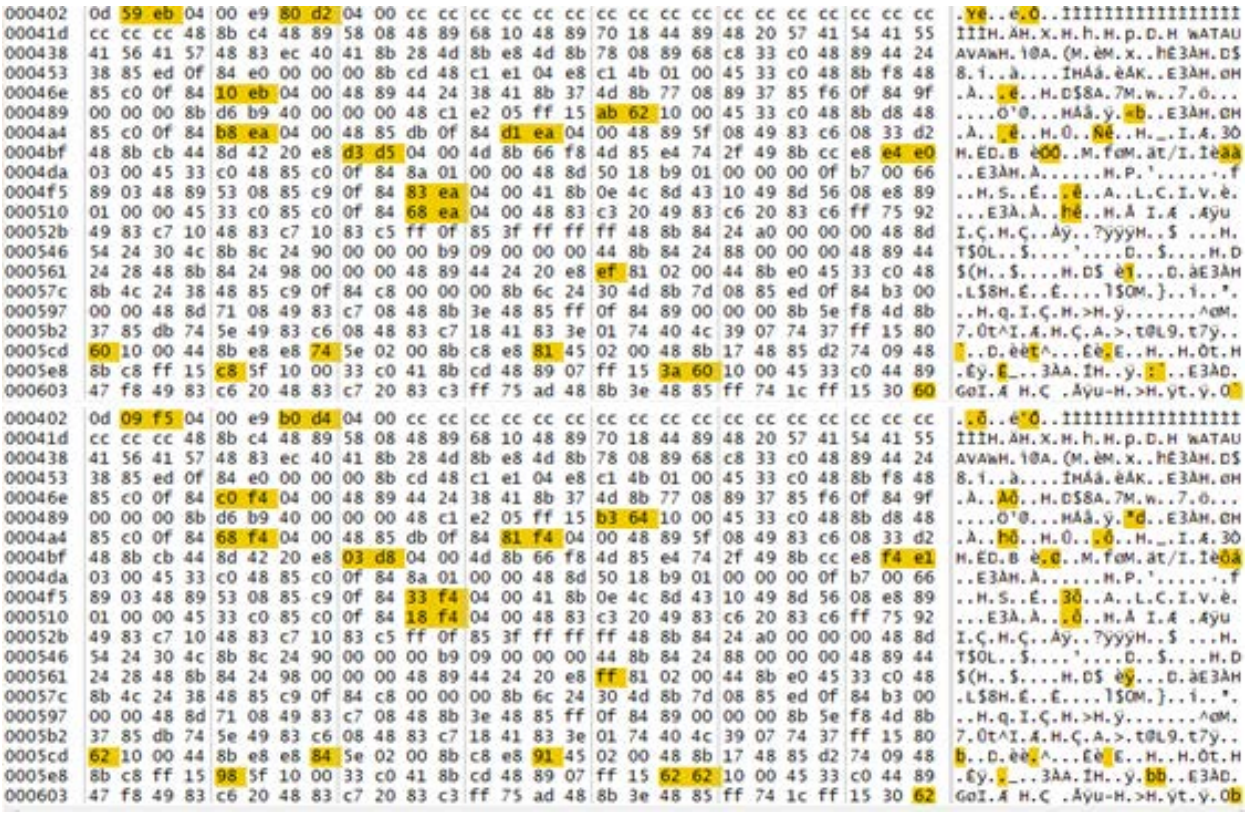

Fig. 7. Comparison of crypt32.dll binaries (ver. 10.0.18362.476 vs. ver. 10.0.18362.592)

3.5 DLL libraries compared

- crypt32-beforePatch.dll - (ver. 10.0.18362.476, buggy version, does not validate base P)
- crypt32-afterPatch.dll - (ver. 10.0.18362.592, patched version, reportedly verifies P baseline)
- crypt32-20H2.dll - (ver. 10.0.19041.1320, version from the last Win 10 Pro update)

Figure 6 is a size comparison of all three.

An attempt to decompile the libraries in the well-known ".NET Reflector" tool was also unsuccessful. In the technical analysis of the problem [13], the author used the BinDiff tool for file comparison, which handled the comparison and partial decompilation of the binary file more successfully. The author focused on the analysis of the most changed methods ChainGetSubjectStatus() and CertObjectCache:FindKnownStoreFlags() and at the same time on the creation of 5 new functions.

4 Conclusion

In this paper, we analyzed the crypt32.dll error in verifying ECC certificates of CAs or elliptic curve parameters. We discussed and analyzed a critical Windows ecosystem bug CVE-0601-2020, where the algorithm for verifying trusted ECC certificates was incorrectly implemented.

It is clear, that the security of information systems with respect to the use of cryptographic algorithms depends not only on the robustness of the cryptographic algorithm itself, the key length or the computational power of the attacker, but also on the correct implementation of the cryptographic algorithm. An incorrect implementation of a cryptographic algorithm has been demonstrated, for example, in the software of the Sony PlayStation 3 product [9], or, as also shown in this paper, in the implementation of the crypt32.dll library in the Microsoft ecosystem. Subsequently, although Microsoft has released

the 14.01.2020 operating systems update, but in practice there are many un-updated Windows 10, Windows server 2016 and Windows server 2019 installed with the old version of the crypt32.dll library. These systems remain critically vulnerable to cyber-attacks.

Since we were not able to analyze with available tools which part of the code in the crypt32.dll library provides a fix for the critical CVE-0601-2020 vulnerability, in the next part of the work we would like to find a tool that can more successfully decompile the individual libraries. At the same time, we will try to set up a virtual workstation in the Windows ecosystem, where we will further test the crypt32.dll library by spoofing a fake certificate authority certificate.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and Public -Key Cryptosystems", *Communication of the ACM*, vol. 21, No. 2, 1978.

[2] W. Diffie and M. E. Hellman, "New direction in cryptography", *IEEE Trans. Info. Theory*, 1976.

[3] E. Gilbert, F. MacWilliams, and N. Sloane, "Codes, which detect deception", *The Bell System technical Journal*, vol. 53, no. 3, pp. 405-424, 1974.

[4] J. Jonsson and B. Kalisky, "Public-Key Cryptography Standards (PKCS), Fremont", *Internet Engineering Task Force*, Internet Engineering Task Force.

[5] *Understanding PKI: Concepts, Standards, and Deployment Considerations*, Addison-Wesley Professional; 2nd edition (November 6), 2002.

[6] L. C. Washington, *Elliptic Curves Number Theory and Cryptography*, Boca Raton: CRC Press, 2000.

[7] V. G. Martinez, L. H. Encinas, and C. S. Avila, "A Survey of the Elliptic Curve Integrated Encryption Scheme", *Journal of computer science and engineering*, vol 2, ISSUE 2, 2010.

[8] M. Dubyk and R. R. Varuni, *Examining CVE-2020-0601 Crypt 32.dll Elliptic Curve Cryptography (ECC) Certificate Validation Vulnerability*, The SANS institute, 01.03. 2022.

[9] G. FailOverFlow, "Console Hacking - PS3 Epic fails", 2010.

[10] K. Burda, *Aplikovan kryptografie*,Brno, VUTIUM, (in Czech), 2013.

[11] *SEC1: Elliptic Curve Cryptography*, Mnonoauga: Certicom Research, 2000.

[12] *NSA, Patch Critical Cryptographic Vulnerability in Microsoft Windows Clients and Servers*, 14. January, 2020.

[13] J. Simpson, *A technical analysis of CurveBall*, (CVE–0601), February, 2020.

[14] O. Lyak, *POC for CVE–0601 Windows CryptoAPI*, (Crypt32. dll), http, s: //github. com/ly4k/CurveBall, January, 2020.

Received 21 May 2022

Štefan Počarovský Ing, is an external PhD student at the Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava.

Martin Koppl Ing, is pursuing his PhD student at the Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava.

Miloš Orgoň born in Pieštany, Slovakia, in 1956, received the Master degree and PhD degree from the Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava in 1980 and 1988, respectively. He is an associate professor at the Department of Telecommunications of FEI STU Bratislava. He has been engaged in the research and development of telecommunication networks and services in liberalized environment for the area of convergent technologies. At present he is engaged in the research of the optimal design of networks and technological components, implementation of functions, services and applications and data security. xcel- lence for SMART technologies, systems and services.