Examining CVE-2020-0601 Crypt32.dll Elliptic Curve Cryptography (ECC) Certificate Validation Vulnerability

GIAC (GXPN) Gold Certification

Author: Maksim Dubyk, modubyk@gmail.com
Advisor: Rajat Ravinder Varuni
Accepted: March 1th, 2022

© 2022 The SANS Institute, Author Retains Full Rights

Abstract

Since its inception, asymmetric cryptography use cases include public key encryption, the construction of a secure channel to share a session key, and the act of digital signing. X.509 is the most prevalent standard for public key certificates in internet protocols. While compatible with a diverse set of algorithms and parameters, the use of Elliptic Curve Cryptography (ECC) for public keys has become a modern standard due to its security utility. As with any capability targeted for abuse by motivated and capable threat actors, the use of ECC public keys in X.509 certificates have dependencies that are often a valuable target. One such software dependency is the mechanism for trusted certificate validation. In Windows operating system environments, the CryptoAPI often performs this function. This paper will explore the critical vulnerability CVE-2020-0601 in crypt32.dll that enables an attacker to forge trusted ECC-based X.509 public key certificates without knowing a valid private key. In addition to a technical exploration of this certificate Validation Vulnerability, this paper proposes several detection methodologies and practical detection with Yara rules.

© 2022 The SANS Institute Author retains full rights.

1. Introduction

Asymmetric cryptography, also known as public key cryptography, originated in the 1970s to alleviate the difficulties of multiple entities sharing and agreeing on a single symmetric key distributed through insecure communication channels (Paar, Pelzl, 2010). Before the advent of asymmetric cryptography, two parties would often leverage unsophisticated and even in-person methods to disseminate a shared symmetric key. As the number of participants in a cryptographic system increases, this challenge only compounds. Asymmetric cryptography utilizes two keys to achieve the desired end. One key is publicly available and referred to as the public key. The other key is assumed to be kept entirely private from the public. This key pair enables users, systems, and applications to establish confidential communication channels through the publicly available key.

Public key cryptography is often made possible through mathematical computations involving prime numbers. The most widely adopted implementation is the RSA (Rivest-Shamir-Adleman) algorithm. A prime number is any number greater than one that is not a product of two smaller natural numbers. In its simplest form, asymmetric cryptography leverages two large prime numbers that are multiplied together and codified in a public key. As intended with all cryptographic trapdoor functions, this operation is easy to process or compute in one direction; however, it is challenging to obtain the original components. This is achieved by the ease in multiplying two prime numbers and the difficulty of factoring the product to identify the actual component prime numbers. It is a challenging computation for an attacker to factor the resulting product (Paar, Pelzl, 2010).

Alice will obtain Bob's public key in the typical example of asymmetric cryptography. Alice will then encrypt her plaintext message with Bob's public key. Alice can then send this encrypted message to Bob through an insecure or untrusted communication channel. Because Alice encrypted the message with Bob's public key, only the corresponding private key, a part of the same key pair, can successfully decrypt the message. Bob receives the message and uses his private key to decrypt the message.

© 2022 The SANS Institute Author retains full rights.

Public key cryptography also gave way to digital signing capabilities. A digital signature is the computation of a hash digest that is encrypted with the signer's private key. The entity validating the signature can use the signer's public key to decrypt the encrypted digest. After computing the same hash digest of the underlying content, the recipient can validate both digests are equal. These operations ensure the integrity of the underlying content and simultaneously authenticate the signer. While signing verification can authenticate the certificate issuer that performed a digital signing, it does not indicate whether the verifier has any trust relationship with the issuer (Kennedy, 2021). This is often addressed by implementing a Public Key Infrastructure (PKI) to define a hierarchy of trust through at least one Certificate Authority (CA). In the modern Windows ecosystem, a typical system leverages a combination of internal and external CAs to verify identities and issue certificates. Windows leverages certificate stores to establish a Certificate Trust List (CTL) of CAs and certificates where a trusted relationship has been established. At the time of this writing, a typical Windows environment has a CTL that is comprised of 514 trusted Certificate Authorities (Microsoft Common CA Knowledge Database, 2020). The combination of public key cryptography and trust hierarchies are core to modern computing and enable transparent and trusted transactions.

2. Elliptic Curve Cryptography (ECC)

The strength of a private and public key pair often depends on the mathematical properties of increasing difficulty when factoring a large integer comprised of several prime factors. While this principle has become the de-facto standard for public key cryptography in the form of RSA, there exists a need to increase the resources required for an attacker to determine the original two prime components. As readily available computational resources increase and provide attackers additional capacity to operationalize resource-intensive mathematical computations, a subsequent increase in key size is typically invoked as defenders' response. However, long-term, this is not a sustainable path as mobile and other devices have limited computational resources.

Elliptic curves have arisen as an alternative to the computational rigor in factoring prime numbers due to the utility in achieving the desired aim for a cryptographic trapdoor

© 2022 The SANS Institute Author retains full rights.

function. An elliptic curve is an algebraic curve of genus one and is defined over a finite field. Elliptic curves are non-singular curves lacking cusps or self-intersections.

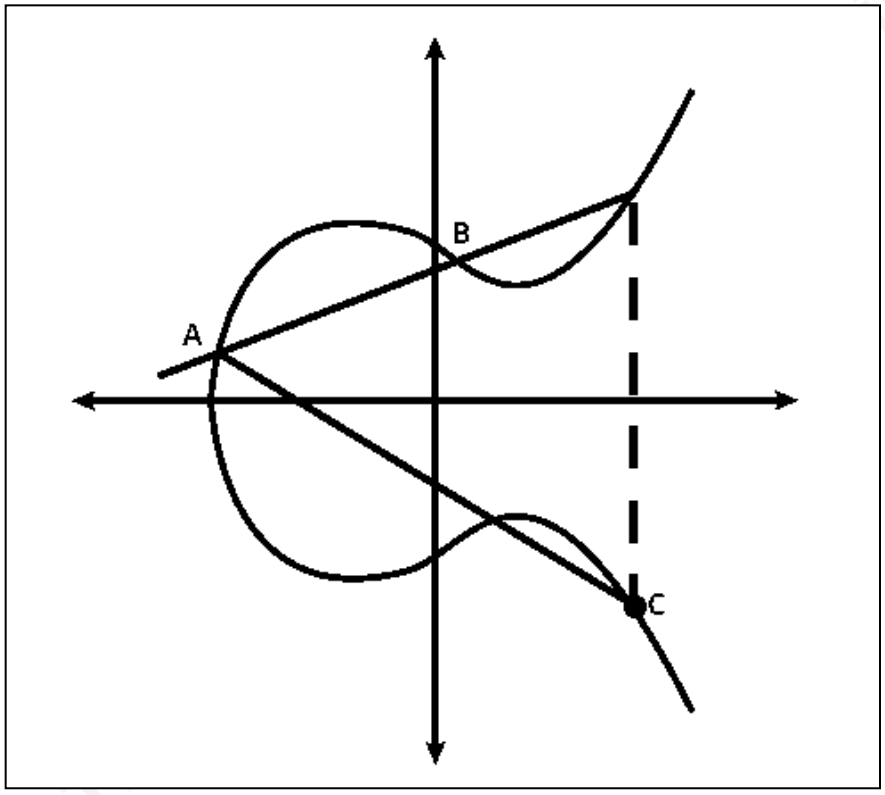


Figure 1. Example Elliptic Curve (Sullivan, 2019).

This plain algebraic curve consists of points (x, y) for the following equation (Koblitz, Menezes, & Vanstone, 2000).

$$y^2 = x^3 + ax + b$$

As depicted in Figure 1, elliptic curves are horizontally symmetrical. A single point on the given curve can be inverted over the x-axis and exist at a different location on the same elliptic curve (Hankerson et al., 2006). Furthermore, a single non-vertical line may meet the elliptic curve at 0 or up to 3 instances. These properties allow an elliptic curve to exhibit behavior that gives way to the elliptic curve discrete logarithm function (Hankerson, Menezes, & Vanstone, 2006).

© 2022 The SANS Institute Author retains full rights.

Elliptic curves share a standard set of parameters that define any given curve's equation and individual properties. These curve features are commonly called elliptic curve domain parameters and are defined as a sextuple over F_p (Certicom Research, 2000). The sextuple of domain parameters is defined as *(p, a, b, G, n, h)* .

The first parameter, *p*, is a prime integer specifying a finite field F_p. Coefficients *a* and *b* in the scope of F_p define the curve equation of $y^2 = x^3 + ax + b$. The Generator *G* (also known as the base point) is a single point on the curve (x_G, y_G). *N* and *h* are the prime order of *G* and cofactor, respectively. The selection of each of these parameters defines an elliptic curve and serves two high-level purposes. First, agreement on curve parameters and subsequent properties enable interoperability between participating parties leveraging elliptic curves for cryptographic ends. For activities such as digital signature verification, results derived from elliptic curve-based computations must be consistent and accurate. In addition, standards on specific curve parameters enable commonly understood security levels. For example, a known named curve with a given set of domain parameters may supply *t* bits of security.

Elliptic Curve Cryptography (ECC) is based on the elliptic curve discrete logarithm function. By selecting elliptic curve domain parameters over a finite field, a secret value acts as a private key. It operates on a single point on the curve, which derives an entirely new second point. This resulting second point codifies the public key. To compute a private key $k_{private}$, a scalar number is selected of modulo *p* (the maximum prime number over the finite field F_p). The public key k_{public} is computed through the multiplication of *G* and $k_{private}$. In the context of elliptic curves, scalar multiplication is implemented through the repeated addition of a point along a curve where *n* is a scalar integer and *P* is an intersection of the curve comprised of *x* and *y* values.

$$nP = P + P + P + P \ldots + P$$

Therefore, the computation of a public key k_{public} in the context of a given elliptic curve can be computed through the following equation.

$$k_{public} = G \cdot k_{private}$$

© 2022 The SANS Institute Author retains full rights.

One can easily obtain the list of standard named elliptic curves compatible in the Microsoft ecosystem by using the Certutil utility (Gerend, 2021).

Curve Name	Curve OID	Key Length	Curve Type
nistP256	1.2.840.10045.3.1.7	256	23
nistP384	1.3.132.0.34	384	24
brainpoolP256r1	1.3.36.3.3.2.8.1.1.7	256	26
brainpoolP384r1	1.3.36.3.3.2.8.1.1.11	384	27
brainpoolP512r1	1.3.36.3.3.2.8.1.1.13	512	28
nistP192	1.2.840.10045.3.1.1	12	19
nistP224	1.3.132.0.33	224	21
nistP521	1.3.132.0.35	521	25
secP160k1	1.3.132.0.9	160	15
secP160r1	1.3.132.0.8	160	16

Figure 2. Microsoft SSL compatible curves obtained via the command *certutil -displayEccCurve* (truncated output)

Certutil can also inspect the domain parameters for an individual curve. As an example, this paper will explore the elliptic curve nistP256 (Cohen, Moody, 2019). Nistp256 has the following elliptic curve domain parameters.

© 2022 The SANS Institute Author retains full rights.

```
C:\>certutil -displayEccCurve nistP256

Curve OID: 1.2.840.10045.3.1.7

Curve Name: nistP256

Public Key Length: 256

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

CertUtil: -displayEccCurve command completed successfully.
```

Figure 3. NistP256 curve parameters obtained from the command *certutil -DisplayEccCurve*.

The base point G is shown in the uncompressed encoding format of elliptic curve points. This method of recording G will be relevant when examining ECC-based X.509 certificates that exploit CVE-2020-0601. The uncompressed encoding P_U is defined as the concatenation of a prefix value 0x04, x_G, and y_G ($C \,||\, X \,||\, Y$) (Technical guideline TR-03111 Elliptic Curve Cryptography, 2012).

© 2022 The SANS Institute Author retains full rights.

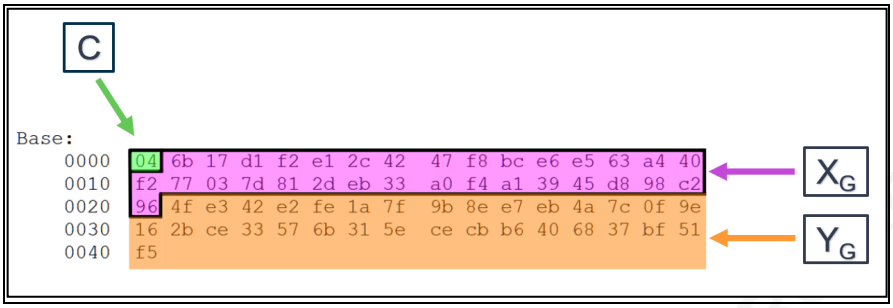


Figure 4. Annotated uncompressed encoding of elliptic curve base point *G* for the named curve nistP256.

As with all asymmetric key cryptography or trapdoor functions, this resulting public key k_{public} is easy to compute, but mathematically it is computationally resource-intensive to identify the private key $k_{private}$ used as a multiplicand with the base point *G*. However, it is very fast to compute k_{public} by multiplying *G* and $k_{private}$. One such efficient elliptic curve scalar multiplication algorithm is the Double-and-add (Eisenträger, Lauter, & Montgomery, 2002). Given two points on an elliptic curve $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, $2P + Q$ can be computed by first adding $P + Q$ compute (x_3, y_3), where

$$\lambda_1 = (y_2 - y_1) / (x_2 - x_1)$$

$$x_3 = \lambda^2_1 - x_1 - x_2, \text{ and}$$

$$y_3 = (x_1 - x_3)\, \lambda_1 - y_1.$$

This result of (x_3, y_3) would then be added to *P* to compute the final value of (x_4, y_4) by

$$\lambda_2 = (y_3 - y_1) / (x_3 - x_1), \text{ and}$$

$$x_4 = \lambda^2_2 - x_1 - x_3, \text{ and}$$

$$y_4 = (x_1 - x_4)\, \lambda_2 - y_1.$$

ECC brought several advantages compared to the widely adopted and de-facto RSA standard. While algorithms such as Shor's have demonstrated utility in the prime factorization problem core to RSA's security, algorithms such as Baby-Step Giant-Step and Pollard rho have yet to reduce the full exponential time required to solve the elliptic curve discrete logarithm problem (expected running time of $\sqrt{\pi n/2}$) (Mahto, Yadav, 2017). In practice, this allows ECC to leverage smaller key sizes than RSA to achieve

© 2022 The SANS Institute Author retains full rights.

comparable levels of security. According to NIST, 256 bits of security can be obtained by an RSA key size of 15,360 bits, whereas ECC requires a significantly smaller key length of 512 bits (Mahto, Yadav, 2017).

Security Bit Level	RSA	ECC
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Figure 5. NIST recommended Security Bit Level (Mahto, Yadav, 2017).

One study explored the level of security offered by various cryptographic ecosystems through an approximation of the volume of water that could be brought to a boiling temperature based on the energy required to break a cryptographic algorithm (Lenstra, Kleinjung, & Thom, 2013). This study found that the energy needed to break a 228-bit RSA key amounted to less energy than was necessary to boil a single teaspoon of water. However, the energy required to break a 228-bit elliptic curve key amounted to energy sufficient to boil all the water on earth.

Due to the increased security achieved through equivalent key sizes, ECC often incurs less overhead and may require reduced computational resources. When comparing the time needed to complete encryption and decryption of 64 bits at a 128-security bit level, RSA took 46.6 seconds, whereas ECC only 22.4 seconds (Mahto, Yadav, 2017). ECC continues to have increased security utility and shows promise for systems with constrained or limited resources.

3. Examining CVE-2020-0601

3.1. CVE-2020-0601 Vulnerability Overview

© 2022 The SANS Institute Author retains full rights.

On January 14, 2020, the National Security Agency (NSA) released a cyber security advisory identifying a critical vulnerability in the Windows operating system (DoD, 2020). CVE-2020-0601 was identified as a flaw in the certificate validation capability within the Windows cryptographic ecosystem. The vulnerability gave a path for attackers to forge X.509 certificates in such a way that Windows could incorrectly validate them as signed by legitimately trusted entities ("An In-Depth Technical Analysis of CurveBall (CVE-2020-0601)", 2020).

The ramifications of successful exploitation were highly significant. Two immediate desirable use cases for attackers emerged. First, an attacker could craft a certificate to sign executable code that could be validated as trusted by the underlying Windows operating system. This is significant because an attacker's malicious content could be validated as trusted and executed, defeating many modern operating system-based defensive measures. The second immediate attack path leveraging this vulnerability is the possibility of HTTPS connections appearing trusted due to a forged X.509 certificate. Modern internet browsers like Google Chrome, Safari, Edge, Firefox, and others, validate TLS certificates against a store of trusted entities. An attacker could exploit CVE-2020-0601 so that the Windows operating system validates the forged certificate as legitimately signed by one of those trusted entities. This would allow an arbitrary untrusted website to appear safe and trusted during the client's TLS certificate validation.

CVE-2020-0601 is rooted in a flaw within the dynamically linked library (DLL) at the core of the Microsoft Windows cryptographic ecosystem, often referred to as the CryptoAPI. crypt32.dll is the module that implements most Windows Certificate and Cryptographic functions, including the acts of both signing and verification (RPW, 2020). The crypt32.dll function ChainGetSubjectStatus is a core component in the certificate validation process for Windows. This specific function gave way to CVE-2020-0601 by an error in the certificate validation process. When presented with a Certificate Authority's (CA) certificate, the CryptoAPI leveraged the issuer and trusted root public key hash to find an equivalent certificate in the trusted system certificate store ("An In-Depth Technical Analysis of CurveBall (CVE-2020-0601)", 2020). However, no

© 2022 The SANS Institute Author retains full rights.

additional comparison was made between the elliptic curve parameters of the certificate undergoing validation and the trusted CA certificate already in the system store. As a result, a certificate undergoing validation could have modified ECC parameters different from those of the equivalent trusted certificate and still be considered trusted. With the ability to modify elliptic curve domain parameters, an adversary is presented with the opportunity of crafting a trusted CA certificate without knowledge of a CA's private key $k_{private}$. This validation flaw eliminates the enforcement of one of the primary assumptions already mentioned in ECC methodology.

The elliptic curve discrete logarithm function is rooted in a publicly known and agreed-upon starting point on a given curve (the base point G). The difficulty in reversing ECC cryptography is, in part, based on the assumption of this shared starting point when operated on by a secret value $k_{private}$ that derives a second point on the curve k_{public}. Suppose an attacker knows the base point G and k_{public}. In that case, it is mathematically complex to identify the $k_{private}$ value that was scalar multiplied with G for a given set of curve parameters. In CVE-2020-0601, the function ChainGetSubjectStatus fundamentally changes the difficulty in the elliptic curve discrete logarithm function by allowing any base point G to be chosen, rather than the agreed-upon standard for a given elliptic curve. If a forged certificate can specify any arbitrary base point, then the elliptic curve discrete logarithm function is significantly easier to solve and identify the $k_{private}$ value that acted upon the base point. Recall the original calculation for public key computation.

$$k_{public} = G \cdot k_{private}$$

Given crypt32.dll does not enforce the agreed-upon G for a trusted certificate, an attacker can now experiment with selecting a custom G to satisfy the requirement for an unknown $k_{private}$. To exploit CVE-2020-0601, an attacker can craft a trusted CA certificate by generating an arbitrary $k_{private}$ value and compute a G that can satisfy the k_{public} of a CA certificate already trusted. To do so, one must take the inverse of the $k_{private}$ as a multiplicand with k_{public} such as the following.

$$G = k_{private}^{-1} \cdot k_{public}$$

© 2022 The SANS Institute Author retains full rights.

As a result, it is trivial to trick crypt32.dll to validate a CA certificate without knowledge of $k_{private}$ due to the ability for an attacker to select an arbitrary $k_{private}$ value and subsequently compute a valid base point *G*. After an attacker successfully crafts this forged CA certificate with modified ECC parameters it can then be leveraged like any other trusted CA certificate to complete certificate signing requests (CSRs).

This vulnerability resulted in an updated patch for the crypt32.dll library. Microsoft remediated CVE-2020-0601 by adding logic to ensure that a certificate undergoing validation is not permitted to specify a non-standard or new base point that differs from the actual existing trusted certificate. This was implemented through the addition of a comparison between the parameters and bytes in a trusted root certificate and the certificate undergoing trust validation ("An In-Depth Technical Analysis of CurveBall (CVE-2020-0601)", 2020). Therefore, signature verification will fail if an attacker modifies the base point parameter for a given elliptic curve to differ from the actual trusted root certificate.

This flaw was confirmed by independent security researchers who isolated the CVE-2020-0601 patch from Microsoft to identify the additional code remediating this vulnerability (RPW, 2020). The vulnerable function ChainGetSubjectStatus was modified to include a newly created function call to ChainComparePublicKeyParametersAndBytes ("An In-Depth Technical Analysis of CurveBall (CVE-2020-0601)", 2020). This function performs the comparison described above.

```
v29 = ChainComparePublicKeyParametersAndBytes(
        *((int **)v5 + 47),
        *((int **)v5 + 48),
        (int *)&v8->pCertInfo->SubjectPublicKeyInfo.Algorithm.Parameters,
        (__int64)&v8->pCertInfo->SubjectPublicKeyInfo.PublicKey);
if ( v29 <= 0 )
{
LABEL_50:
  if ( v29 )
  {
    v7->dwErrorStatus |= 8u;
    *v4 &= 0xFFFFFEFF;
  }
  goto LABEL_39;
}
if ( !CryptVerifyCertificateSignatureEx(0i64, 1u, 2u, pvSubject, 2u, v8, 0, 0i64) )
{
  ChainLogMSRC54294Error(v8, *((_QWORD *)v5 + 47));
  goto LABEL_50;
}
```

Figure 6. Decompiled CVE-2020-0601 crypt32.dll patch showing a new function call ChainComparePublicKeyParametersAndBytes that verifies ECC parameters (RPW, 2020).

Maksim Dubyk, modubyk@gmail.com

© 2022 The SANS Institute Author retains full rights.

3.2. ECC codified in X.509 Certificates

To examine what successful exploitation of CVE-2020-0601 will look like in practice, one first must understand how an X.509 certificate is expected to be structured and formatted. After this is understood, one can then examine how an X.509 certificate can be crafted to specifically leverage the lack of ECC curve parameter verification in the crypt32.dll CVE-2020-0601 vulnerability.

X.509 is a cryptographic standard for defining the format and structure of public key certificates. Through a digital signature, X.509 certificates tie a specific identity to a public key. Digital certificates include the version number, serial number, signature algorithm ID, Issuer name, validity period, subject name, subject public key info, and other optional details. Each X.509 certificate can be self-signed or signed by a known certificate authority. ECC-based public keys are identified by the presence of an algorithm Object ID (OID) value of 1.2.840.10045.2.1 (ecPublicKey). Figure 7 demonstrates the high-level structure of a typical X.509 certificate for an ECC-based public key.

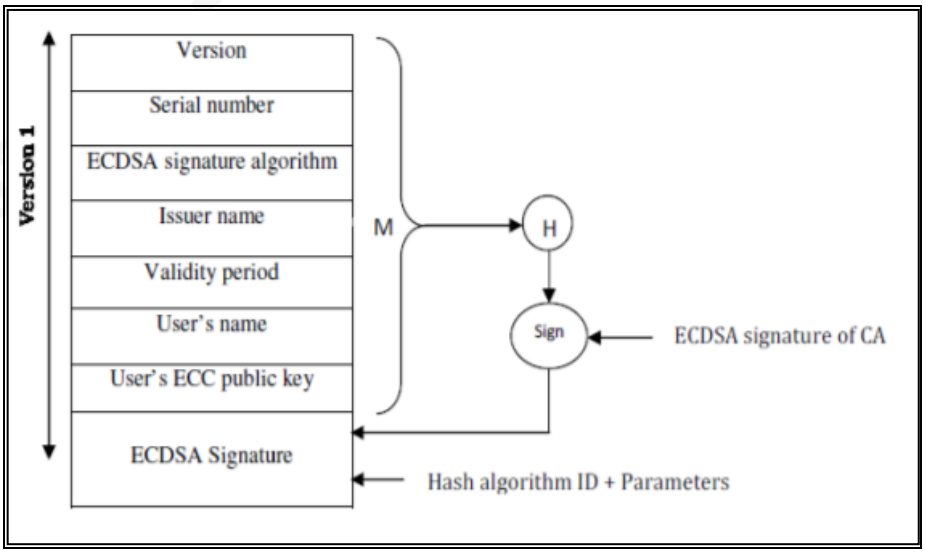


Figure 7. X.509 structure for ECC-based public key certificate (Ray, Biswas, 2013).

X.509 certificates are codified in Abstract Syntax Notation One (ASN.1). While ASN.1 is the standard for X.509 notation, Distinguished Encoding Rules (DER) is the encoded output. When examining X.509 certificates, one is almost always dealing with

© 2022 The SANS Institute Author retains full rights.

DER-encoded ASN.1 notation, or the PEM base64 encoded equivalent. DER can be described as a "type-length-value" encoding scheme. In this encoding format, the content is codified in hexadecimal representation, where a type indicates a specific value type (integer, string, sequence, etc.) followed by the content's length. Lastly, the actual content follows the length. Figure 8 provides an example of how the DER-encoded ASN.1 can be understood for a sample X.509 certificate.

```
tag length data                                | comment
=== ====== ================================= | ========================================================
 30 820335                                     | ASN.1 header
---------------------------------------------- | --------------------------------------------------------
 30 82021D                                     | ''to be signed'' part begins here
 A0 03                                         |
 02 01     02                                  | X.509 version 3
 02 04     03507449                            | serial number (0x03507449)
 30 0D                                         |
 06 09     2A864886F70D010104                  | signature algorithm identifier (md5withRSAEncryption)
 05 00                                         |
```

Figure 8. Example ASN.1 DER-encoded X.509 certificate (Lenstra, Wang, & de Weger, 2005).

As with any public key algorithm, supporting details must be captured in the X.509 certificate for usage and verification of a given public key certificate. IETF RFC 3279 standardizes how elliptic curve details are recorded in an X.509 certificate (RFC3279, 2002). When an X.509 certificate contains an algorithm OID of 1.2.840.10045.2.1 (ecPublicKey) specifying the use of ECC, the algorithm's parameters are defined in the EcpkParameters structure.

```
EcpkParameters ::= CHOICE {
  ecParameters  ECParameters,
  namedCurve    OBJECT IDENTIFIER,
  implicitlyCA  NULL }
```

Figure 9. EcpkParameters ASN.1 structure definition for elliptic curve algorithm details (RFC3279, 2002).

The EcpkParameters structure is defined as an ASN.1 data of type "choice," indicating the structure may contain only one of the entities listed in its definition. This definition can be understood as a certificate explicitly defining its elliptic curve domain

© 2022 The SANS Institute Author retains full rights.

parameters (ecParameters), identifying the OID of a known named curve (namedCurve), or a NULL value implicitly inheriting elliptic curve algorithm details from a Certificate Authority (implicitlyCA). As previously discussed, to successfully exploit CVE-2020-0601, a specially crafted certificate must provide custom and non-standard elliptic curve domain parameters. The ECParameters structure is an ASN.1 sequence data type constructed of multiple ordered fields of different data types (Figure 10). This structure includes the elliptic curve domain parameters that can be defined in the context of an X.509 certificate.

```
ECParameters ::= SEQUENCE {
  version   ECPVer,          -- version is always 1
  fieldID   FieldID,         -- identifies the finite field over
                             -- which the curve is defined
  curve     Curve,           -- coefficients a and b of the
                             -- elliptic curve
  base      ECPoint,         -- specifies the base point P
                             -- on the elliptic curve
  order     INTEGER,         -- the order n of the base point
  cofactor  INTEGER OPTIONAL -- The integer h = #E(Fq)/n
  }
```

Figure 10. ECParameters ASN.1 sequence defining elliptic curve domain parameters (RFC3279, 2002).

To examine this sequence, a sample X.509 certificate with the Serial Number f0:cf:fa:f4:86:ae:a5:99:0a:00:00:00:01:10:37:9b was collected from www.google.com. Using the Certutil utility, one can extract and parse the ASN.1 from the certificate to identify details such as the signer identity, public key, cryptographic algorithm, and other supporting artifacts. Of note, the Object ID (OID) value of 1.2.840.10045.2.1 identifies this certificate as containing an elliptic curve-based public key. Furthermore, the EcpkParameters structure contains the OID 1.2.840.10045.3.1.7 that identifies a named curve prime256v1 or secp256r1.

© 2022 The SANS Institute Author retains full rights.

```
0097:    |  30 19                          ; SEQUENCE (19 Bytes)
0099:    |  |  31 17                       ; SET (17 Bytes)
009b:    |  |     30 15                    ; SEQUENCE (15 Bytes)
009d:    |  |        06 03                 ; OBJECT_ID (3 Bytes)
009f:    |  |        |  55 04 03
         |  |        |     ; 2.5.4.3 Common Name (CN)
00a2:    |  |        13 0e                 ; PRINTABLE_STRING (e Bytes)
00a4:    |  |           77 77 77 2e 67 6f 6f 67  6c 65 2e 63 6f 6d        ; www.google.com
         |  |              ; "www.google.com"
00b2:    |  30 59                          ; SEQUENCE (59 Bytes)
00b4:    |  |  30 13                       ; SEQUENCE (13 Bytes)
00b6:    |  |     06 07                    ; OBJECT_ID (7 Bytes)
00b8:    |  |     |  2a 86 48 ce 3d 02 01                                  ecPublicKey Object ID (OID)
         |  |     |     ; 1.2.840.10045.2.1 ECC
00bf:    |  |     06 08                    ; OBJECT_ID (8 Bytes)
00c1:    |  |        2a 86 48 ce 3d 03 01 07                               prime256v1 Object ID (OID)
         |  |           ; 1.2.840.10045.3.1.7 ECDSA_P256 (x962P256v1)
00c9:    |  |  03 42                       ; BIT_STRING (42 Bytes)
00cb:    |  |     00
00cc:    |  |     04 03 18 11 e6 d2 71 79  8a e9 5b 1b e0 58 d1 f9
00dc:    |  |     7a 9e 72 72 06 90 02 ef  f2 d4 53 1c 2d 58 a4 36
00ec:    |  |     65 55 e4 1a 20 e3 15 93  cc 2d 75 97 8c 8a d5 40
00fc:    |  |     49 1f 2b fb 08 ab 26 4f  c2 7a 74 86 67 58 bd 33
010c:    |  |     a4
```

Figure 11. Annotated ASN.1 of Google certificate to identify ecPublicKey OID and EcpkParameters structure identifying the named curve secp256r1.

3.3. X.509 Certificates Crafted to Exploit CVE-2020-0601

Within days of the NSA Advisory and Microsoft patch being released, publicly available proof-of-concept exploit code emerged. Security researcher Oliver Lyak ("Ollypwn") released 13 lines of ruby code demonstrating the ease at which one could forge X.509 certificates to exploit CVE-2020-0601 (Lyak, 2020). This proof-of-concept code computed a valid base point *G* for a given public key k_{public} by making a straightforward choice. Lyak's exploit code set the private key $k_{private}$ equal to one. As previously reviewed, the public key can be computed under normal circumstances by the following equation.

$$k_{public} = G \cdot k_{private}$$

However, with $k_{private}$ set to a value of one, the equation now becomes two equal points on the curve.

$$k_{public} = G$$

This made it trivial to spoof a CA's public key certificate without knowing the true private key and ultimately result in the certificate validated by crypt32.dll as trusted.

© 2022 The SANS Institute Author retains full rights.

A sample was obtained from Oliver's Github repository (Lyak, 2020). Lyak forged a certificate with the same public key as the Microsoft ECC Product Root Certificate Authority in this proof-of-concept exploit. By default, this root CA is included in the Windows 10 trusted certificate store and thus would be a valuable choice for an attacker to spoof. Figure 12 shows partial output when using Openssl to view certificate details for the legitimate ECC Product Root Certificate Authority certificate and has a serial number of 14:98:26:66:dc:7c:cd:8f:40:53:67:7b:b9:99:ec:85. Of note, it leverages ECC cryptography and the named curve secp384r1.

```
$ openssl x509 -in MicrosoftECCProductRootCertificateAuthority.cer -noou
t -text -inform PEM
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            14:98:26:66:dc:7c:cd:8f:40:53:67:7b:b9:99:ec:85
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C = US, ST = Washington, L = Redmond, O = Microsoft Corporation, CN = Mi
crosoft ECC Product Root Certificate Authority 2018
        Validity
            Not Before: Feb 27 20:42:08 2018 GMT
            Not After : Feb 27 20:50:46 2043 GMT
        Subject: C = US, ST = Washington, L = Redmond, O = Microsoft Corporation, CN = M
icrosoft ECC Product Root Certificate Authority 2018
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:c7:11:16:2a:76:1d:56:8e:be:b9:62:65:d4:c3:
                    ce:b4:f0:c3:30:ec:8f:6d:d7:6e:39:bc:c8:49:ab:
                    ab:b8:e3:43:78:d5:81:06:5d:ef:c7:7d:9f:ce:d6:
                    b3:90:75:de:0c:b0:90:de:23:ba:c8:d1:3e:67:e0:
                    19:a9:1b:86:31:1e:5f:34:2d:ee:17:fd:15:fb:7e:
                    27:8a:32:a1:ea:c9:8f:c9:7e:18:cb:2f:3b:2c:48:
                    7a:7d:a6:f4:01:07:ac
                ASN1 OID: secp384r1
                NIST CURVE: P-384
        X509v3 extensions:
```

Figure 12. X.509 certificate details of the legitimate Microsoft ECC Product Root Certificate Authority certificate.

Lyak's proof-of-concept certificate was explicitly crafted to appear as this same trusted root CA. Figure 13 shows the partial DER-encoded ASN.1 notation for this forged certificate that was obtained with the *cerutil -asn* command. Viewing this ASN.1 notation reveals several interesting attributes. While the legitimate certificate's EcpkParameters structure had the 1.3.132.0.34 OID value indicating the use of the named curve secp384r1, the forged certificate instead has the ECParameters structure at offset

© 2022 The SANS Institute Author retains full rights.

0x00F0, indicating the presence of explicitly provided elliptic curve domain parameters. The presence of curve domain parameters is also apparent by the OID 1.2.840.10045.1.1 (prime-field) located at offset 0x00FB, indicating an ASN sequence including the prime integer that specifies a finite field F_p for the elliptic curve. The ECParameters sequence structure at offset 0x00F0 defines all the parameters *p, a, b, G, n,* and *h* introduced in Section 2.

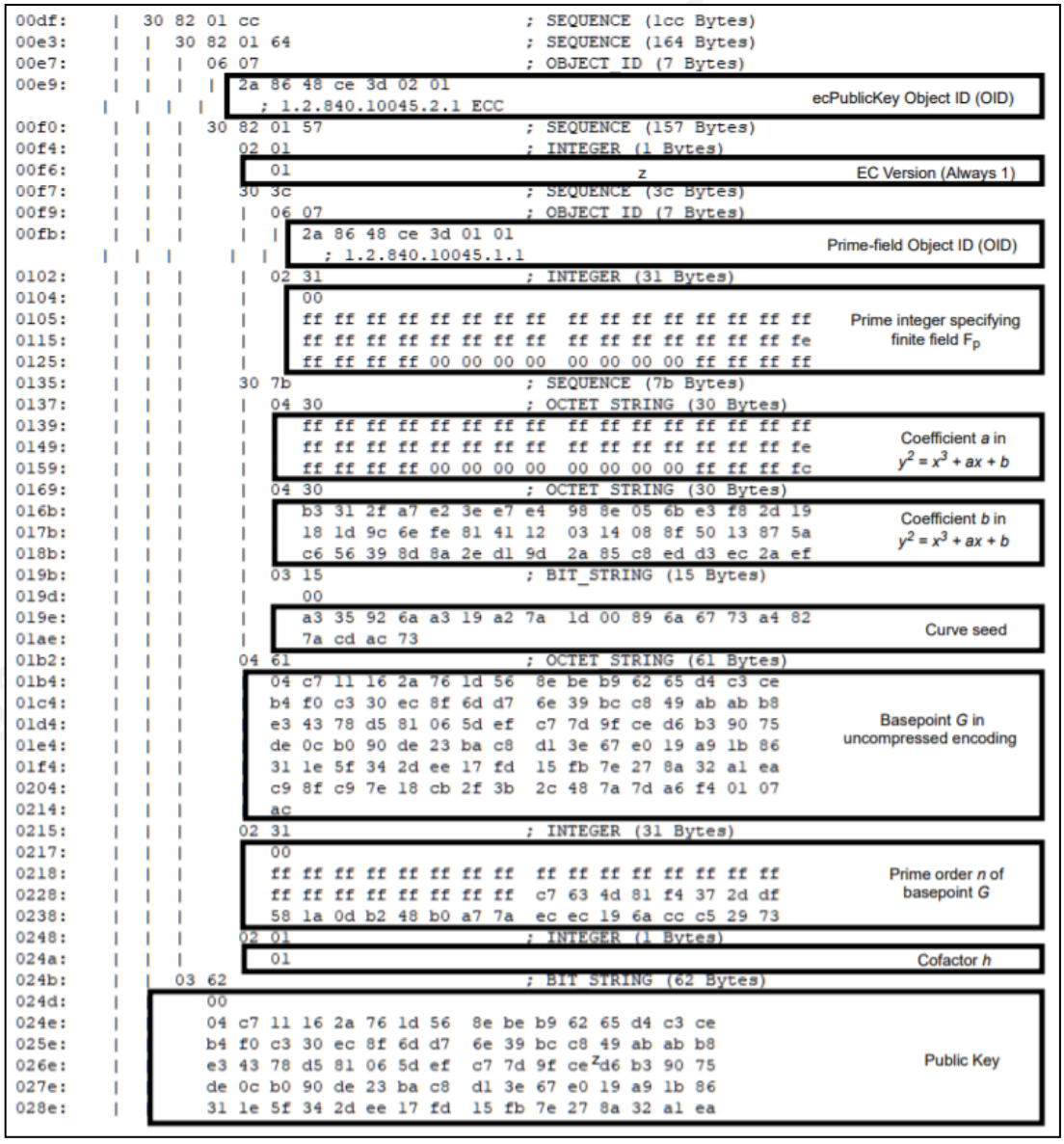


Figure 13. Annotated ASN.1 of Lyak's CVE-2020-0601 PoC certificate

A comparison of each parameter in the forged certificate with that of the standard parameters for the named curve secp384r1 reveals only one that differs, the base point *G*.

© 2022 The SANS Institute Author retains full rights.

The standard base point *G* for the named curve secp384r1 can be obtained with the certutil command *certutil -displayEccCurve secp384r1* and provides the following *G* value in hexadecimal form which differs from the value located at offset 0x01B4 in Figure 13.

```
0000  04 aa 87 ca 22 be 8b 05  37 8e b1 c7 1e f3 20 ad
0010  74 6e 1d 3b 62 8b a7 9b  98 59 f7 41 e0 82 54 2a
0020  38 55 02 f2 5d bf 55 29  6c 3a 54 5e 38 72 76 0a
0030  b7 36 17 de 4a 96 26 2c  6f 5d 9e 98 bf 92 92 dc
0040  29 f8 f4 1d bd 28 9a 14  7c e9 da 31 13 b5 f0 b8
0050  c0 0a 60 b1 ce 1d 7e 81  9d 7a 43 1d 7c 90 ea 0e
0060  5f
```

The presence of the non-standard base point *G* for named curve secp384r1 is one indicator this certificate is attempting to exploit CVE-2020-0601. However, another highly unusual feature of this forged root CA certificate exists. As previously mentioned, Lyak's proof-of-concept exploit code sets the $k_{private}$ value equal to one and has the subsequent effect of causing the base point *G* and k_{public} to be identical. When reviewing the ASN.1 notation for this forged certificate, it is also apparent. The base point G located at offset 0x01B4 and k_{public} located at offset 0x024E have the same uncompressed encoded point value.

While Ollypwn set $k_{private}$ equal to one due to the ease in computing a valid base point *G* (simply setting equal to k_{public}), this was certainly not a requirement to successfully exploit CVE-2020-0601. Researchers at Kudelski Security also released proof-of-concept exploit code where a value of two was instead chosen for $k_{private}$ (Pelissier, 2020). The researchers chose a different trusted CA to spoof in this POC, the USERTrust ECC Certification Authority. This trusted root CA similarly leverages the named curve secp384r1 (Figure 14).

© 2022 The SANS Institute Author retains full rights.

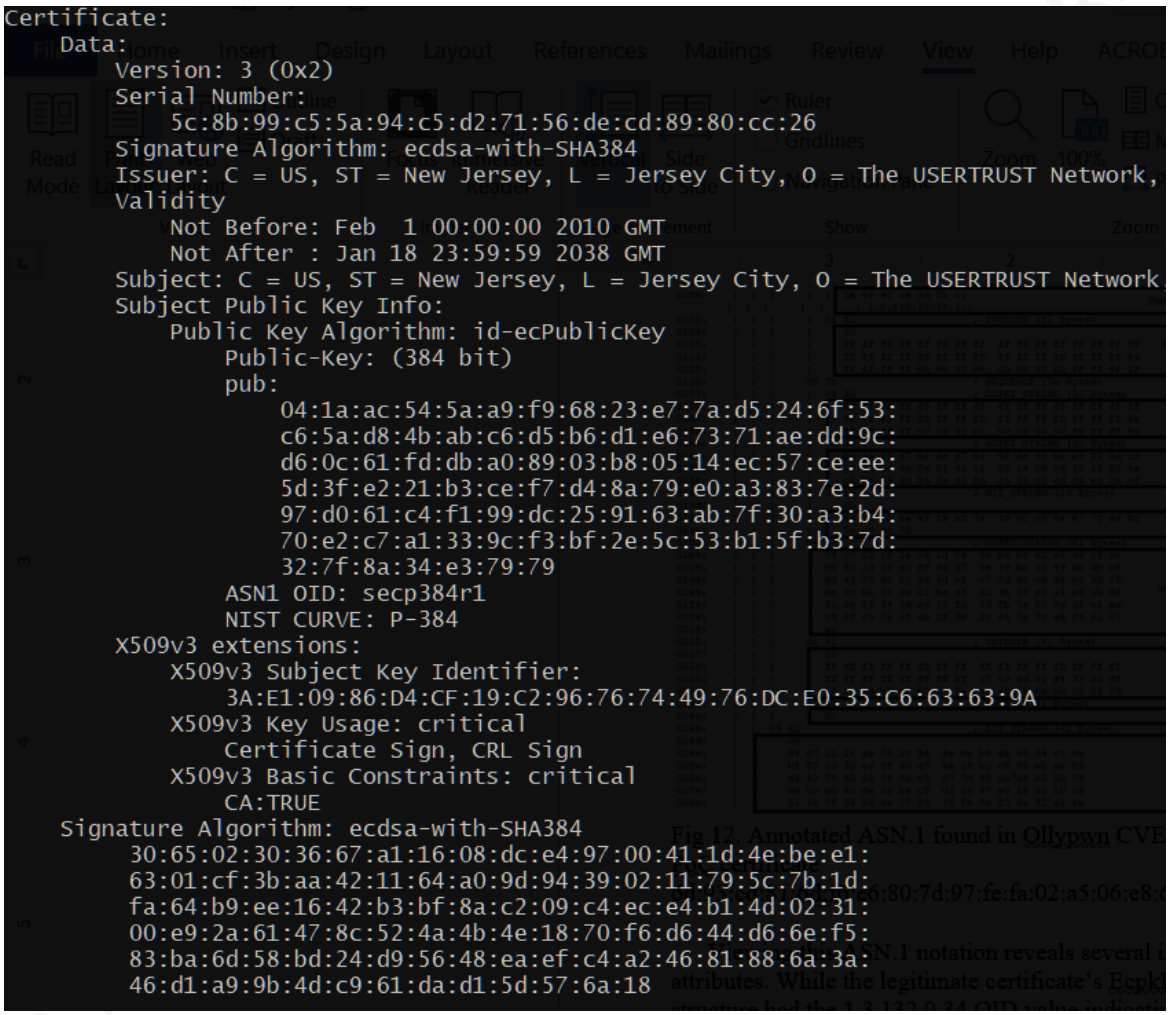


Figure 14. Legitimate USERTrust ECC Certification Authority certificate details.

Like the first certificate examined that exploits CVE-2020-0601, the ASN.1 notation of Kudelski Security's certificate reveals explicitly defined curve domain parameters as suspected (Figure 15). The public key at offset 0x02A9 remains equivalent to the legitimate USERTrust ECC Certification Authority certificate public key. Now turning attention to the base point at offset 0x020F. As suspected, this base point is not the standard *G* value for the named curve secp384r1 indicating this certificate is attempting to exploit CVE-2020-0601. However, different than Lyak's choice of one for $k_{private}$, a value of two was used. Because of this choice, the base point and public key no longer have equal uncompressed encoded point values.

© 2022 The SANS Institute Author retains full rights.

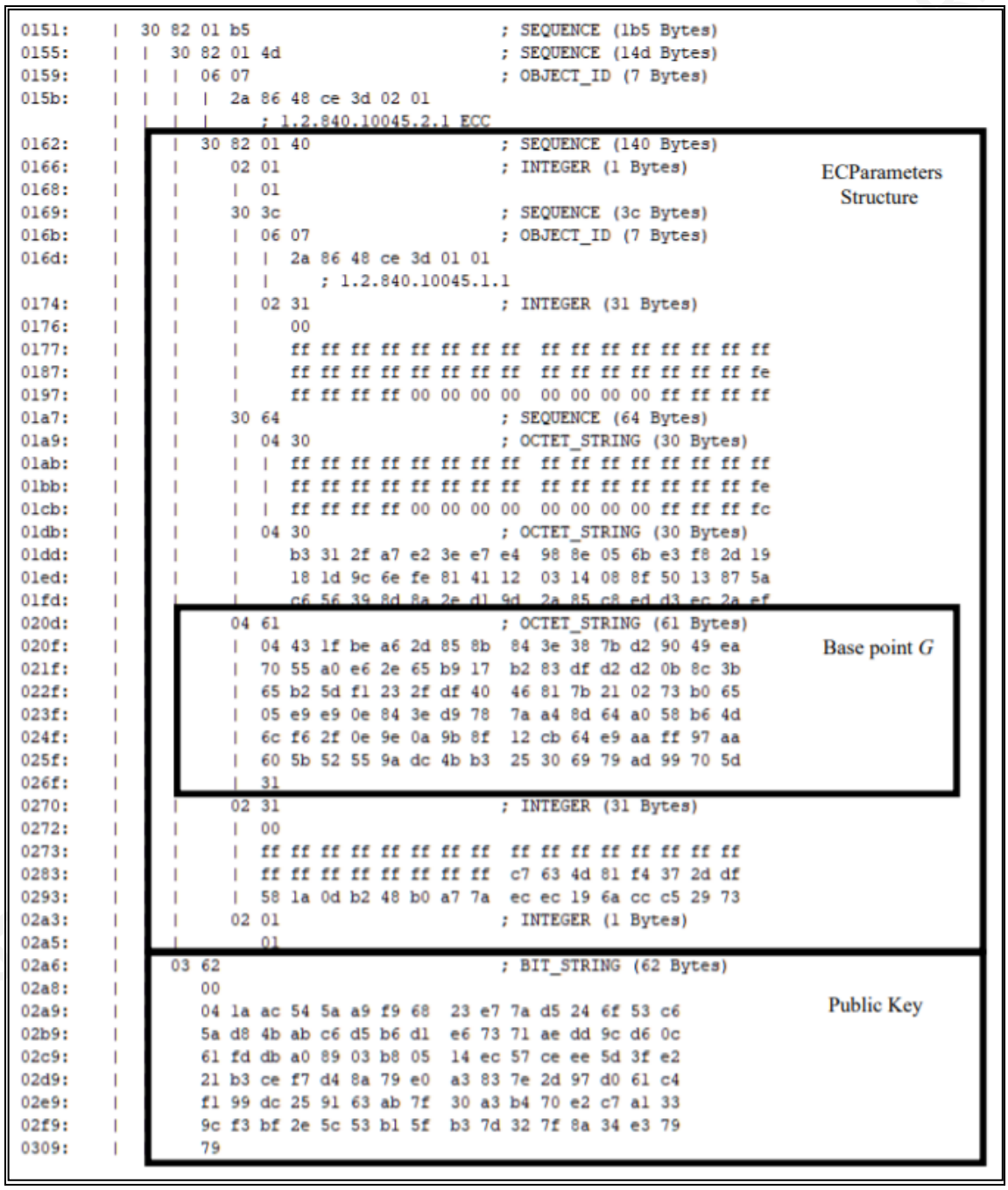


Figure 15. Forged CA Certificate using Kudelski Security CA

4. CVE-2020-0601 Detection Methodologies

While a patch has been released to remediate the flaw in crypt32.dll, it is still a valuable pursuit to explore detection opportunities for defenders. Many systems likely

© 2022 The SANS Institute Author retains full rights.

remain unpatched. Furthermore, it is a worthwhile exercise to explore how one may leverage an understanding of CVE-2020-0601 in the broader context of elliptic curve cryptography principles to detect certificates crafted for exploitation. This section will explore several detection methodologies and how they may be implemented in Yara. Complete copies of the Yara rules presented in this section can be found in Appendix A and Github (Dubyk, 2022). Each rule was written to operate on DER-encoded X.509 certificate content. The proposed rules were written for research and hunting purposes and have not been validated to run in production environments.

The first detection methodology to consider is the presence of explicitly defined ECC domain parameters within X.509 certificates. This approach does not detect any specific parameters that may be unique to CVE-2020-0601 exploitation; however, the presence of these parameters altogether is uncommon and anomalous by itself. The OID 1.2.840.10045.1.1 (prime-field) can be used to identify the ASN.1 structure ECParameters that are indicative of explicitly defined elliptic curve domain parameters. Implementing this criterion in Yara can be achieved by the following rule.

```
/* Sample DER encoded ASN.1
0159:    |  |  |  06 07                        ; OBJECT_ID (7 Bytes)
015b:    |  |  |  |  2a 86 48 ce 3d 02 01
                                 |  |  |  |     ; 1.2.840.10045.2.1 ECC
0162:    |  |  |  30 82 01 40                  ; SEQUENCE (140 Bytes)
0166:    |  |  |     02 01                     ; INTEGER (1 Bytes)
0168:    |  |  |     |  01
0169:    |  |  |     30 3c                     ; SEQUENCE (3c Bytes)
016b:    |  |  |     |  06 07                  ; OBJECT_ID (7 Bytes)
016d:    |  |  |     |  |  2a 86 48 ce 3d 01 01
                        |  |  |     |  |     ; 1.2.840.10045.1.1
*/

rule explicitly_defined_curve_domain_parameters
{
    strings:
        $ = { 06 07 2a 86 48 ce 3d 02 01 30 82 [2] 02 01 01 30 3c 06 07 2a 86 48 ce 3d 01 01 }

    condition: all of them
}
```

Figure 16. Yara rule to identify ECC-based X.509 certificates with explicitly defined elliptic curve domain parameters.

While this Yara rule acts as a starting point to detect anomalous X.509 certificates where elliptic curve domain parameters are explicitly defined, there are several use cases where this criterion will be met in certificates that are benign and not attempting to exploit CVE-2020-0601. For example, it's possible the ECParameters structure is being used to define curve attributes for a standard named curve. Relying solely on the presence

© 2022 The SANS Institute Author retains full rights.

of explicitly defined parameters will not be sufficient to detect certificates exploiting CVE-2020-0601 with accuracy. Instead, one must consider a more targeted approach.

As commonly observed in other exploitation techniques, an adversary's choices can result in downstream anomalies or observables that may result in detection opportunities. This is precisely the case with the first proof-of-concept certificate previously examined. Lyak's choice of setting the private key $k_{private}$ to one resulted in equal values for the public key k_{public} and base point G, which should not occur under normal circumstances. Implemented in Figure 17, the proposed detection methodology identifies X.509 certificates where the public key k_{public} and base point G are equal. This is achieved by first identifying explicitly defined curve domain parameters as already implemented in the prior Yara rule. In addition, this rule locates the relative offsets where the public key k_{public} and base point G begin. Lastly, it will iterate over each byte of the public key k_{public} and base point G for the complete key size and require equal values.

```
rule CVE_2020_0601_basePoint_equal_publicKey {

strings:

    $ecc_custom_params = { 06 07 2a 86 48 ce 3d 02 01 30 82 [2] 02 01 01 30 3c 06 07 2a 86 48 ce 3d 01 01 }

condition:

    for any x in (1..#ecc_custom_params):
        (for all i in
            (0..
                // Public Key Size
                uint8be(
                    @ecc_custom_params[1] // start at ecPublicKey OID
                    + uint16be(@ecc_custom_params[1] + 11)  // jump 11 bytes and resolve ecParameters sequence length tag
                    + 13 // jump 13 bytes to ecParameters sequence value content
                    + 1 // jump 1 bytes to bit string length tag
                ) - 3 // subtract 3 bytes (1 because iterator starts at 0, 1 for bit string unused bits header (0x00), 1 for unused byte
            ) :
                (
                    uint16be(
                        @ecc_custom_params[1] // start at ecPublicKey OID
                        + uint16be(@ecc_custom_params[1] + 11)  // jump 11 bytes and resolve ecParameters sequence length tag
                        + 13 // jump 13 bytes to ecParameters sequence value content
                        + 3 // jump 3 bytes past bit string type length header to value content
                        + i
                    )
                    ==
                    uint16be(
                        @ecc_custom_params[1] // start at ecPublicKey OID
                        + 20 // jump 20 bytes to FieldID value content
                        + uint8be(@ecc_custom_params[1] + 17) // resolve and jump FieldID object Length
                        + ( // resolve and jump Curve object length
                            uint8be (
                                @ecc_custom_params[1] // start at ecPublicKey OID
                                + 19 // jump 19 bytes to FieldID length tag
                                + uint8be(@ecc_custom_params[1] + 17)  // resolve FieldID object Length
                            )
                        )
                        + 2 // jump 2 bytes past octet string type length header to value content
                        + i
                    )

                )
        )
}
```

Figure 17. Yara rule to identify ECC-based X.509 certificates with explicitly defined elliptic curve domain parameters where the G and k_{public} values are equal.

© 2022 The SANS Institute Author retains full rights.

While the previous rule is one high fidelity means of identifying CVE-2020-0601 exploitation, the choice of setting $k_{private}$ equal to one is not a requirement to forge a trusted CA that could be validated by the flaw in crypt32.dll. This was demonstrated by the proof-of-concept certificate where $k_{private}$ was equal to two, resulting in the public key k_{public} and base point G having different values (Pelissier, 2020). Additionally, with varying key sizes and flexibility offered in X.509 ASN.1 notation, it is unlikely the previously proposed rule will be comprehensive enough to provide adequate detection coverage. For these reasons, a more robust detection methodology is needed.

Leveraging an understanding of the requirements needed to exploit CVE-2020-0601, a more comprehensive approach emerges. An explicitly defined custom base point G value different from those of the standard named curves used in Windows is an accurate means to detect certificates exploiting CVE-2020-0601. This detection approach also accounts for differences in an adversary's choice of the private key $k_{private}$ value. This detection logic can be codified in a Yara rule by extending the first rule initially proposed. In addition to identifying explicitly defined curve parameters, the rule will ensure the G value is not a known base point for the standard named curves. The sample rule in Figure 18 implements this criterion in a Yara rule. The uncompressed encoded G values were truncated for easier reading of the screenshot. The complete Yara rule can be found in Appendix A. Of note, the condition section of the Yara rule requires explicitly defined ECC parameters to exist in the certificate. Simultaneously, this rule enforces that none of the G values found in standard named curves are found.

© 2022 The SANS Institute Author retains full rights.

```
rule cve_2020_0601_ECC_non_standard_basePoint
{
    strings:

        $ecc_custom_params = { 06 07 2a 86 48 ce 3d 02 01 30 82 [2] 02 01 01 30 3c 06 07 2a 86 48 ce 3d 01 01 }

        $basePoint_secp384r1 = { 04 aa 87 ca 22 be 8b 05 37 8e b1 c7 1e     ... }
        $basePoint_secP256r1 = {04 6b 17 d1 f2 e1 2c 42 47 f8 bc e6 e5 63   ... }
        $basePoint_curve25519 = { 04 00 00 00 00 00 00 00 00 00 00 00 00    ... }
        $basePoint_nistP256 = { 04 6b 17 d1 f2 e1 2c 42  47 f8 bc e6 e5 63  ... }
        $basePoint_nistP384 = { 04 aa 87 ca 22 be 8b 05 37 8e b1 c7 1e f3   ... }
        $basePoint_brainpoolP256r1 = { 04 8b d2 ae b9 cb 7e 57 cb 2c 4b 48  ... }
        $basePoint_brainpoolP384r1 = { 04 1d 1c 64 f0 68 cf 45 ff a2 a6 3a  ... }
        $basePoint_brainpoolP512r1 = { 04 81 ae e4 bd d8 2e d9 64 5a 21 32  ... }
        $basePoint_nistP192 = { 04 18 8d a8 0e b0 30 90 f6 7c bf 20 eb 43   ... }
        $basePoint_nistP224 = { 04 b7 0e 0c bd 6b b4 bf 7f 32 13 90 b9 4a   ... }
        $basePoint_nistP521 = { 04 00 c6 85 8e 06 b7 04 04 e9 cd 9e 3e cb   ... }
        $basePoint_secP160k1 = { 04 3b 4c 38 2c e3 7a a1  92 a4 01 9e 76 30 ... }
        $basePoint_secP160r1 = { 04 4a 96 b5 68 8e f5 73  28 46 64 69 89 68 ... }
        $basePoint_secP160r2 = { 04 52 dc b0 34 29 3a 11 7e 1f 4f f1 1b 30  ... }
        $basePoint_secP192k1 = { 04 db 4f f1 0e c0 57 e9 ae 26 b0 7d 02 80  ... }
        $basePoint_secP192r1 = { 04 18 8d a8 0e b0 30 90 f6 7c bf 20 eb 43  ... }
        $basePoint_secP224k1 = { 04 a1 45 5b 33 4d f0 99 df 30 fc 28 a1 69  ... }
        $basePoint_secP224r1 = { 04 b7 0e 0c bd 6b b4 bf 7f 32 13 90 b9 4a  ... }
        $basePoint_secP256k1 = { 04 79 be 66 7e f9 dc bb ac 55 a0 62 95 ce  ... }
        $basePoint_secP384r1 = { 04 aa 87 ca 22 be 8b 05 37 8e b1 c7 1e f3  ... }
        $basePoint_secP521r1 = { 04 00 c6 85 8e 06 b7 04 04 e9 cd 9e 3e cb  ... }
        $basePoint_brainpoolP160r1 = { 04 be d5 af 16 ea 3f 6a 4f 62 93 8c  ... }
        $basePoint_brainpoolP160t1 = { 04 b1 99 b1 3b 9b 34 ef  c1 39 7e 64 ... }
        $basePoint_brainpoolP192r1 = { 04 c0 a0 64 7e aa b6 a4 87 53 b0 33  ... }
        $basePoint_brainpoolP192t1 = { 04 3a e9 e5 8c 82 f6 3c 30 28 2e 1f  ... }
        $basePoint_brainpoolP224r1 = { 04 0d 90 29 ad 2c 7e 5c f4 34 08 23  ... }
        ...
        ...
        ...

condition:
    $ecc_custom_params
    and not any of ($basePoint*)
}
```

Figure 18. Yara rule to identify ECC-based X.509 certificates with explicitly defined non-standard base point *G* parameters (truncated values for easier reading)

5. Conclusion

CVE-2020-0601 was a critical ECC certificate validation vulnerability in the Windows ecosystem. It emphasized the necessity for not only sound cryptographic methods, but the validation performed by downstream utilities such as crypt32.dll must likewise be sound. A perfect encryption scheme guaranteeing infinity security bits fails to be successful if its integrity fails to be verified by a flawed approach like what was observed in CVE-2020-0601.

Furthermore, exploring the world of elliptic curve cryptography can quickly become mathematically very technical. However, CVE-2020-0601 reveals that a firm foundation in the core principles of asymmetric cryptography and elliptic curves can go a long way in practical application. One does not need to be an expert in elliptic curves or the discrete logarithm problem to understand CVE-2020-0601 and identify possible detection approaches. As the state of information technology only becomes more

© 2022 The SANS Institute Author retains full rights.

interconnected with a greater reliance on cryptography, defenders will need to continue growing their understanding to keep up with the dynamic nature of threats carried out by capable and motivated threat actors.

© 2022 The SANS Institute, Author Retains Full Rights

Maksim Dubyk, modubyk@gmail.com

© 2022 The SANS Institute Author retains full rights.

References

"An In-Depth Technical Analysis of CurveBall (CVE-2020-0601)." TrendLabs Security Intelligence Blog, 13 Feb. 2020, blog.trendmicro.com/trendlabs-security-intelligence/an-in-depth-technical-analysis-of-curveball-cve-2020-0601/,

Certicom Research. (2000, September 20). Recommended elliptic curve domain parameters. Standards for Efficient Cryptography (SEC). Retrieved November 27, 2021, from https://www.secg.org/SEC2-Ver-1.0.pdf.

Cohen, L., & Moody, D. (2019, October). Recommendations for discrete logarithm-based ... - NIST. Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters. Retrieved November 13, 2021, from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf.

DoD. (2020, January 14). Patch Critical Cryptographic Vulnerability in Microsoft Windows Clients and Servers. Depart Of Defense Media. Retrieved November 6, 2021, from https://media.defense.gov/2020/Jan/14/2002234275/-1/-1/0/CSA-WINDOWS-10-CRYPT-LIB-20190114.PDF.

Dubyk, M. (2022, February 8). *modubyk/CVE_2020_0601*. GitHub. Retrieved February 9, 2022, from https://github.com/modubyk/CVE_2020_0601

Eisenträger, K., Lauter, K., & Montgomery, P. L. (2002). An efficient procedure to double and add points on an elliptic curve. Cryptology ePrint Archive, Report 2002/112.

Gerend, J. (n.d.). Certutil. Microsoft Docs. Retrieved November 13, 2021, from https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/certutil.

Hankerson, D., Menezes, A. J., & Vanstone, S. (2006). Guide to elliptic curve cryptography. Springer Science & Business Media.

Kennedy, J. (2021, July 1). *Hierarchy of Trust*. Microsoft Windows App Development. Retrieved December 31, 2021, from https://docs.microsoft.com/en-us/windows/win32/seccrypto/hierarchy-of-trust

Koblitz, N., Menezes, A., & Vanstone, S. (2000). The state of elliptic curve cryptography. Designs, codes and cryptography, 19(2), 173-193.

Maksim Dubyk, modubyk@gmail.com

© 2022 The SANS Institute Author retains full rights.

Lenstra, A. K., Wang, X., & de Weger, B. M. M. (2005). Colliding X. 509 certificates (No. REP_WORK).

Lenstra, A. K., Kleinjung, T., & Thomé, E. (2013). Universal security. In Number theory and cryptography (pp. 121-124). Springer, Berlin, Heidelberg.

Lyak, O. (2020, January 15). Ly4k/Curveball: POC for CVE-2020-0601- Windows CryptoAPI (Crypt32.dll). GitHub. Retrieved November 30, 2021, from https://github.com/ly4k/CurveBall.

Mahto, D., & Yadav, D. K. (2017). RSA and ECC: a comparative analysis. International journal of applied engineering research, 12(19), 9053-9061.

Microsft. (2021, December 31). *Microsoft Included CA certificate list*. Microsoft Common CA Knowledge Database (CCADB). Retrieved December 31, 2021, from https://ccadb-public.secure.force.com/microsoft/IncludedCACertificateReportForMSFT

Paar, C., & Pelzl, J. (2010). Introduction to public-key cryptography. In Understanding Cryptography (pp. 149-171). Springer, Berlin, Heidelberg.

Pelissier, S. (2020, January 15). Kudelskisecurity/Chainoffools: A POC for CVE-2020-0601. GitHub. Retrieved November 30, 2021, from https://github.com/kudelskisecurity/chainoffools.

Ray, S., & Biswas, G. P. (2013). Design of mobile public key infrastructure (M-PKI) using elliptic curve cryptography. International Journal on Cryptography and Information Security (IJCIS), 3(1), 25-37.

RFC3279 Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Internet Engineering Task Force (IETF). (2002, April). Retrieved December 5, 2021, from https://datatracker.ietf.org/doc/html/rfc3279.

(RPW), R. (2020, January 14). Ok, this explains the call to chaincomparepublickeyparametersandbytes() in chaingetsubjectstatus(): Pic.twitter.com/73kigqtgw9. Twitter. Retrieved November 10, 2021, from https://twitter.com/esizkur/status/1217162360072425478.

Sullivan, Nick. "A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography." The Cloudflare Blog, The Cloudflare Blog, February 15 2019,

© 2022 The SANS Institute Author retains full rights.

blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/.

Technical guideline TR-03111 Elliptic Curve Cryptography. Bundesamt fur Sicherheit in der Informationstechnik. (2012, June 28). Retrieved November 27, 2021, from https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03111/BSI-TR-03111_V-2-1_pdf.pdf?__blob=publicationFile&v=2.

6. Appendix A – Yara Rules

6.1. Explicitly defined elliptic curve domain parameters

```
rule explicitly_defined_curve_domain_parameters
{
    strings:
        $ = { 06 07 2a 86 48 ce 3d 02 01 30 82 [2] 02 01 01 30 3c 06 07 2a 86
48 ce 3d 01 01 }

    condition: all of them
}
```

6.2. Explicitly defined elliptic curve domain parameters where the *G* and k_{public} values are equal

```
rule cve_2020_0601_basePoint_equal_publicKey {

strings:

    $ecc_custom_params = { 06 07 2a 86 48 ce 3d 02 01 30 82 [2] 02 01 01 30 3c
06 07 2a 86 48 ce 3d 01 01 }

condition:

    for any x in (1..#ecc_custom_params):
        (for all i in
            (0..
                // Public Key Size
                uint8be(
                    @ecc_custom_params[1] // start at ecPublicKey OID
                    + uint16be(@ecc_custom_params[1] + 11)  // jump 11 bytes
and resolve ecParameters sequence length tag
                    + 13 // jump 13 bytes to ecParameters sequence value
content
                    + 1 // jump 1 bytes to bit string length tag
```


© 2022 The SANS Institute Author retains full rights.

```
                ) - 3 // subtract 3 bytes (1 because iterator starts at 0, 1
for bit string unused bits header (0x00), 1 for unused byte
            ) :
                (
                    uint16be(
                        @ecc_custom_params[1] // start at ecPublicKey OID
                        + uint16be(@ecc_custom_params[1] + 11)  // jump 11
bytes and resolve ecParameters sequence length tag
                        + 13 // jump 13 bytes to ecParameters sequence value
content
                        + 3 // jump 3 bytes past bit string type length header
to value content
                        + i
                    )
                    ==
                    uint16be(
                        @ecc_custom_params[1] // start at ecPublicKey OID
                        + 20 // jump 20 bytes to FieldID value content
                        + uint8be(@ecc_custom_params[1] + 17) // resolve and
jump FieldID object Length
                        + ( // resolve and jump Curve object length
                            uint8be (
                                @ecc_custom_params[1] // start at ecPublicKey
OID
                                + 19 // jump 19 bytes to FieldID length tag
                                + uint8be(@ecc_custom_params[1] + 17)  //
resolve FieldID object Length
                            )
                        )
                        + 2 // jump 2 bytes past octet string type length
header to value content
                        + i
                    )

                )
        )
}
```

6.3. Explicitly defined elliptic curve domain parameters and a non-standard base point *G* parameter

```
rule cve_2020_0601_ECC_non_standard_basePoint
{
strings:

    $ecc_custom_params = { 06 07 2a 86 48 ce 3d 02 01 30 82 [2] 02 01 01 30 3c
06 07 2a 86 48 ce 3d 01 01 }

    $basePoint_secp384r1 = { 04 aa 87 ca 22 be 8b 05 37 8e b1 c7 1e f3 20 ad
74 6e 1d 3b 62 8b a7 9b 98 59 f7 41 e0 82 54 2a 38 55 02 f2 5d bf 55 29 6c 3a
54 5e 38 72 76 0a b7 36 17 de 4a 96 26 2c 6f 5d 9e 98 bf 92 92 dc 29 f8 f4 1d
bd 28 9a 14 7c e9 da 31 13 b5 f0 b8 c0 0a 60 b1 ce 1d 7e 81 9d 7a 43 1d 7c 90
ea 0e 5f}
```


© 2022 The SANS Institute Author retains full rights.

```
    $basePoint_secP256r1 = {04 6b 17 d1 f2 e1 2c 42 47 f8 bc e6 e5 63 a4 40 f2
77 03 7d 81 2d eb 33 a0 f4 a1 39 45 d8 98 c2 96 4f e3 42 e2 fe 1a 7f  9b 8e e7
eb 4a 7c 0f 9e 16 2b ce 33 57 6b 31 5e  ce cb b6 40 68 37 bf 51 f5 }
    $basePoint_curve25519 = { 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 09 20 ae 19 a1 b8 a0 86 b4 e0
1e dd 2c 77 48 d1 4c 92 3d 4d 7e 6d 7c 61 b2 29 e9 c5 a2 7e ce d3 d9 }
    $basePoint_nistP256 = { 04 6b 17 d1 f2 e1 2c 42  47 f8 bc e6 e5 63 a4 40
f2 77 03 7d 81 2d eb 33  a0 f4 a1 39 45 d8 98 c2 96 4f e3 42 e2 fe 1a 7f  9b
8e e7 eb 4a 7c 0f 9e 16 2b ce 33 57 6b 31 5e  ce cb b6 40 68 37 bf 51 f5 }
    $basePoint_nistP384 = { 04 aa 87 ca 22 be 8b 05 37 8e b1 c7 1e f3 20 ad 74
6e 1d 3b 62 8b a7 9b  98 59 f7 41 e0 82 54 2a 38 55 02 f2 5d bf 55 29 6c 3a 54
5e 38 72 76 0a b7 36 17 de 4a 96 26 2c 6f 5d 9e 98 bf 92 92 dc 29 f8 f4 1d bd
28 9a 14 7c e9 da 31 13 b5 f0 b8 c0 0a 60 b1 ce 1d 7e 81 9d 7a 43 1d 7c 90 ea
0e 5f }
    $basePoint_brainpoolP256r1 = { 04 8b d2 ae b9 cb 7e 57 cb 2c 4b 48 2f fc
81 b7 af b9 de 27 e1 e3 bd 23 c2 3a 44 53 bd 9a ce 32 62 54 7e f8 35 c3 da c4
fd 97 f8 46 1a 14 61 1d c9 c2 77 45 13 2d ed 8e 54 5c 1d 54 c7 2f 04 69 97 }
    $basePoint_brainpoolP384r1 = { 04 1d 1c 64 f0 68 cf 45 ff a2 a6 3a 81 b7
c1 3f 6b 88 47 a3 e7 7e f1 4f e3 db 7f ca fe 0c bd 10 e8 e8 26 e0 34 36 d6 46
aa ef 87 b2 e2 47 d4 af 1e 8a be 1d 75 20 f9 c2 a4 5c b1 eb 8e 95 cf d5 52 62
b7 0b 29 fe ec 58 64 e1 9c 05 4f f9 91 29 28 0e 46 46 21 77 91 81 11 42 82 03
41 26 3c 53 15}
    $basePoint_brainpoolP512r1 = { 04 81 ae e4 bd d8 2e d9 64 5a 21 32 2e 9c
4c 6a 93 85 ed 9f 70 b5 d9 16 c1 b4 3b 62 ee f4 d0 09 8e ff 3b 1f 78 e2 d0 d4
8d 50 d1 68 7b 93 b9 7d 5f 7c 6d 50 47 40 6a 5e 68 8b 35 22 09 bc b9 f8 22 7d
de 38 5d 56 63 32 ec c0 ea bf a9 cf 78 22 fd f2 09 f7 00 24 a5 7b 1a a0 00 c5
5b 88 1f 81 11 b2 dc de 49 4a 5f 48 5e 5b ca 4b d8 8a 27 63 ae d1 ca 2b 2f a8
f0 54 06 78 cd 1e 0f 3a d8 08 92 }
    $basePoint_nistP192 = { 04 18 8d a8 0e b0 30 90 f6 7c bf 20 eb 43 a1 88 00
f4 ff 0a fd 82 ff 10 12 07 19 2b 95 ff c8 da 78 63 10 11 ed 6b 24 cd d5 73 f9
77 a1 1e 79 48 11 }
    $basePoint_nistP224 = { 04 b7 0e 0c bd 6b b4 bf 7f 32 13 90 b9 4a 03 c1 d3
56 c2 11 22 34 32 80 d6 11 5c 1d 21 bd 37 63 88 b5 f7 23 fb 4c 22 df e6 cd 43
75 a0 5a 07 47 64 44 d5 81 99 85 00 7e 34 }
    $basePoint_nistP521 = { 04 00 c6 85 8e 06 b7 04 04 e9 cd 9e 3e cb 66 23 95
b4 42 9c 64 81 39 05 3f b5 21 f8 28 af 60 6b 4d 3d ba a1 4b 5e 77 ef e7 59 28
fe 1d c1 27 a2 ff a8 de 33 48 b3 c1 85 6a 42 9b f9 7e 7e 31 c2 e5 bd 66 01 18
39 29 6a 78 9a 3b c0 04 5c 8a 5f b4 2c 7d 1b d9 98 f5 44 49 57 9b 44 68 17 af
bd 17 27 3e 66 2c 97 ee 72 99 5e f4 26 40 c5 50 b9 01 3f ad 07 61 35 3c 70 86
a2 72 c2 40 88 be 94 76 9f d1 66 50 }
    $basePoint_secP160k1 = { 04 3b 4c 38 2c e3 7a a1  92 a4 01 9e 76 30 36 f4
f5 dd 4d 7e bb 93 8c f9  35 31 8f dc ed 6b c2 82 86 53 17 33 c3 f0 3c 4f ee }
    $basePoint_secP160r1 = { 04 4a 96 b5 68 8e f5 73  28 46 64 69 89 68 c3 8b
b9 13 cb fc 82 23 a6 28  55 31 68 94 7d 59 dc c9 12 04 23 51 37 7a c5 fb 32 }
    $basePoint_secP160r2 = { 04 52 dc b0 34 29 3a 11 7e 1f 4f f1 1b 30 f7 19
9d 31 44 ce 6d fe af fe f2 e3 31 f2 96 e0 71 fa 0d f9 98 2c fe a7 d4 3f 2e }
    $basePoint_secP192k1 = { 04 db 4f f1 0e c0 57 e9 ae 26 b0 7d 02 80 b7 f4
34 1d a5 d1 b1 ea e0 6c 7d 9b 2f 2f 6d 9c 56 28 a7 84 41 63 d0 15 be 86 34 40
82 aa 88 d9 5e 2f 9d }
    $basePoint_secP192r1 = { 04 18 8d a8 0e b0 30 90 f6 7c bf 20 eb 43 a1 88
00 f4 ff 0a fd 82 ff 10 12 07 19 2b 95 ff c8 da 78 63 10 11 ed 6b 24 cd d5 73
f9 77 a1 1e 79 48 11 }
```


© 2022 The SANS Institute Author retains full rights.

```
    $basePoint_secP224k1 = { 04 a1 45 5b 33 4d f0 99 df 30 fc 28 a1 69 a4 67
e9 e4 70 75 a9 0f 7e 65 0e b6 b7 a4 5c 7e 08 9f ed 7f ba 34 42 82 ca fb d6 f7
e3 19 f7 c0 b0 bd 59 e2 ca 4b db 55 6d 61 a5 }
    $basePoint_secP224r1 = { 04 b7 0e 0c bd 6b b4 bf 7f 32 13 90 b9 4a 03 c1
d3 56 c2 11 22 34 32 80 d6 11 5c 1d 21 bd 37 63 88 b5 f7 23 fb 4c 22 df e6 cd
43 75 a0 5a 07 47 64 44 d5 81 99 85 00 7e 34 }
    $basePoint_secP256k1 = { 04 79 be 66 7e f9 dc bb ac 55 a0 62 95 ce 87 0b
07 02 9b fc db 2d ce 28 d9 59 f2 81 5b 16 f8 17 98 48 3a da 77 26 a3 c4 65 5d
a4 fb fc 0e 11 08 a8 fd 17 b4 48 a6 85 54 19 9c 47 d0 8f fb 10 d4 b8 }
    $basePoint_secP384r1 = { 04 aa 87 ca 22 be 8b 05 37 8e b1 c7 1e f3 20 ad
74 6e 1d 3b 62 8b a7 9b 98 59 f7 41 e0 82 54 2a 38 55 02 f2 5d bf 55 29 6c 3a
54 5e 38 72 76 0a b7 36 17 de 4a 96 26 2c 6f 5d 9e 98 bf 92 92 dc 29 f8 f4 1d
bd 28 9a 14 7c e9 da 31 13 b5 f0 b8 c0 0a 60 b1 ce 1d 7e 81 9d 7a 43 1d 7c 90
ea 0e 5f }
    $basePoint_secP521r1 = { 04 00 c6 85 8e 06 b7 04 04 e9 cd 9e 3e cb 66 23
95 b4 42 9c 64 81 39 05 3f b5 21 f8 28 af 60 6b 4d 3d ba a1 4b 5e 77 ef e7 59
28 fe 1d c1 27 a2 ff a8 de 33 48 b3 c1 85 6a 42 9b f9 7e 7e 31 c2 e5 bd 66 01
18 39 29 6a 78 9a 3b c0 04 5c 8a 5f b4 2c 7d 1b d9 98 f5 44 49 57 9b 44 68 17
af bd 17 27 3e 66 2c 97 ee 72 99 5e f4 26 40 c5 50 b9 01 3f ad 07 61 35 3c 70
86 a2 72 c2 40 88 be 94 76 9f d1 66 50 }
    $basePoint_brainpoolP160r1 = { 04 be d5 af 16 ea 3f 6a 4f 62 93 8c 46 31
eb 5a f7 bd bc db c3 16 67 cb 47 7a 1a 8e c3 38 f9 47 41 66 9c 97 63 16 da 63
21 }
    $basePoint_brainpoolP160t1 = { 04 b1 99 b1 3b 9b 34 ef  c1 39 7e 64 ba eb
05 ac c2 65 ff 23 78 ad d6 71 8b 7c 7c 19 61 f0 99 1b 84 24 43 77 21 52 c9 e0
ad }
    $basePoint_brainpoolP192r1 = { 04 c0 a0 64 7e aa b6 a4 87 53 b0 33 c5 6c
b0 f0 90 0a 2f 5c 48 53 37 5f d6 14 b6 90 86 6a bd 5b b8 8b 5f 48 28 c1 49 00
02 e6 77 3f a2 fa 29 9b 8f }
    $basePoint_brainpoolP192t1 = { 04 3a e9 e5 8c 82 f6 3c 30 28 2e 1f e7 bb
f4 3f a7 2c 44 6a f6 f4 61 81 29 09 7e 2c 56 67 c2 22 3a 90 2a b5 ca 44 9d 00
84 b7 e5 b3 de 7c cc 01 c9 }
    $basePoint_brainpoolP224r1 = { 04 0d 90 29 ad 2c 7e 5c f4 34 08 23 b2 a8
7d c6 8c 9e 4c e3 17 4c 1e 6e fd ee 12 c0 7d 58 aa 56 f7 72 c0 72 6f 24 c6 b8
9e 4e cd ac 24 35 4b 9e 99 ca a3 f6 d3 76 14 02 cd }
    $basePoint_brainpoolP224t1 = { 04 6a b1 e3 44 ce 25 ff 38 96 42 4e 7f fe
14 76 2e cb 49 f8 92 8a c0 c7 60 29 b4 d5 80 03 74 e9 f5 14 3e 56 8c d2 3f 3f
4d 7c 0d 4b 1e 41 c8 cc 0d 1c 6a bd 5f 1a 46 db 4c }
    $basePoint_brainpoolP256t1 = { 04 a3 e8 eb 3c c1 cf e7 b7 73 22 13 b2 3a
65 61 49 af a1 42 c4 7a af bc 2b 79 a1 91 56 2e 13 05 f4 2d 99 6c 82 34 39 c5
6d 7f 7b 22 e1 46 44 41 7e 69 bc b6 de 39 d0 27 00 1d ab e8 f3 5b 25 c9 be }
    $basePoint_brainpoolP320r1 = { 04 43 bd 7e 9a fb 53 d8 b8 52 89 bc c4 8e
e5 bf e6 f2 01 37 d1 0a 08 7e b6 e7 87 1e 2a 10 a5 99 c7 10 af 8d 0d 39 e2 06
11 14 fd d0 55 45 ec 1c c8 ab 40 93 24 7f 77 27 5e 07 43 ff ed 11 71 82 ea a9
c7 78 77 aa ac 6a c7 d3 52 45 d1 69 2e 8e e1 }
    $basePoint_brainpoolP320t1 = { 04 92 5b e9 fb 01 af c6 fb 4d 3e 7d 49 90
01 0f 81 34 08 ab 10 6c 4f 09 cb 7e e0 78 68 cc 13 6f ff 33 57 f6 24 a2 1b ed
52 63 ba 3a 7a 27 48 3e bf 66 71 db ef 7a bb 30 eb ee 08 4e 58 a0 b0 77 ad 42
a5 a0 98 9d 1e e7 1b 1b 9b c0 45 5f b0 d2 c3 }
    $basePoint_brainpoolP384t1 = { 04 18 de 98 b0 2d b9 a3 06 f2 af cd 72 35
f7 2a 81 9b 80 ab 12 eb d6 53 17 24 76 fe cd 46 2a ab ff c4 ff 19 1b 94 6a 5f
54 d8 d0 aa 2f 41 88 08 cc 25 ab 05 69 62 d3 06 51 a1 14 af d2 75 5a d3 36 74
7f 93 47 5b 7a 1f ca 3b 88 f2 b6 a2 08 cc fe 46 94 08 58 4d c2 b2 91 26 75 bf
5b 9e 58 29 28 }
```


© 2022 The SANS Institute Author retains full rights.

```
    $basePoint_brainpoolP512t1 = { 04 64 0e ce 5c 12 78 87 17 b9 c1 ba 06 cb
c2 a6 fe ba 85 84 24 58 c5 6d de 9d b1 75 8d 39 c0 31 3d 82 ba 51 73 5c db 3e
a4 99 aa 77 a7 d6 94 3a 64 f7 a3 f2 5f e2 6f 06 b5 1b aa 26 96 fa 90 35 da 5b
53 4b d5 95 f5 af 0f a2 c8 92 37 6c 84 ac e1 bb 4e 30 19 b7 16 34 c0 11 31 15
9c ae 03 ce e9 d9 93 21 84 be ef 21 6b d7 1d f2 da df 86 a6 27 30 6e cf f9 6d
bb 8b ac e1 98 b6 1e 00 f8 b3 32 }
    $basePoint_ec192wapi = { 04 4a d5 f7 04 8d e7 09 ad 51 23 6d e6 5e 4d 4b
48 2c 83 6d c6 e4 10 66 40 02 bb 3a 02 d4 aa ad ac ae 24 81 7a 4c a3 a1 b0 14
b5 27 04 32 db 27 d2 }
    $basePoint_numsP256t1 = { 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0d 7d 0a b4 1e 2a 12 76 db a3
d3 30 b3 9f a0 46 bf be 2a 6d 63 82 4d 30 3f 70 7f 6f b5 33 1c ad ba }
    $basePoint_numsP384t1 = { 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 08 74 9c da ba 13 6c e9 b6 5b d4 47 17 94 aa 61 9d aa 5c 7b
4c 93 0b ff 8e bd 79 8a 8a e7 53 c6 d7 2f 00 38 60 fe ba ba d5 34 a4 ac f5 fa
7f 5b ee }
    $basePoint_numsP512t1 = { 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 7d 67 e8
41 dc 4c 46 7b 60 50 91 d8 08 69 21 2f 9c eb 12 4b f7 26 97 3f 9f f0 48 77 9e
1d 61 4e 62 ae 2e ce 50 57 b5 da d9 6b 7a 89 7c 1d 72 79 92 61 13 46 38 75 0f
4f 0c b9 10 27 54 3b 1c 5e }
    $basePoint_wtls7 = { 04 52 dc b0 34 29 3a 11 7e 1f 4f f1 1b 30 f7 19 9d 31
44 ce 6d fe af fe f2 e3 31 f2 96 e0 71 fa 0d f9 98 2c fe a7 d4 3f 2e }
    $basePoint_wtls9 = {04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 }
    $basePoint_wtls12 = { 04 b7 0e 0c bd 6b b4 bf 7f 32 13 90 b9 4a 03 c1 d3
56 c2 11 22 34 32 80 d6 11 5c 1d 21 bd 37 63 88 b5 f7 23 fb 4c 22 df e6 cd 43
75 a0 5a 07 47 64 44 d5 81 99 85 00 7e 34 }
    $basePoint_x962P192v1 = { 04 18 8d a8 0e b0 30 90 f6 7c bf 20 eb 43 a1 88
00 f4 ff 0a fd 82 ff 10 12 07 19 2b 95 ff c8 da 78 63 10 11 ed 6b 24 cd d5 73
f9 77 a1 1e 79 48 11 }
    $basePoint_x962P192v2 = { 04 ee a2 ba e7 e1 49 78 42 f2 de 77 69 cf e9 c9
89 c0 72 ad 69 6f 48 03 4a 65 74 d1 1d 69 b6 ec 7a 67 2b b8 2a 08 3d f2 f2 b0
84 7d e9 70 b2 de 15 }
    $basePoint_x962P192v3 = { 04 7d 29 77 81 00 c6 5a 1d a1 78 37 16 58 8d ce
2b 8b 4a ee 8e 22 8f 18 96 38 a9 0f 22 63 73 37 33 4b 49 dc b6 6a 6d c8 f9 97
8a ca 76 48 a9 43 b0 }
    $basePoint_x962P239v1 = { 04 0f fa 96 3c dc a8 81 6c cc 33 b8 64 2b ed f9
05 c3 d3 58 57 3d 3f 27 fb bd 3b 3c b9 aa af 7d eb e8 e4 e9 0a 5d ae 6e 40 54
ca 53 0b a0 46 54 b3 68 18 ce 22 6b 39 fc cb 7b 02 f1 ae }
    $basePoint_x962P239v2 = { 04 38 af 09 d9 87 27 70 51 20 c9 21 bb 5e 9e 26
29 6a 3c dc f2 f3 57 57 a0 ea fd 87 b8 30 e7 5b 01 25 e4 db ea 0e c7 20 6d a0
fc 01 d9 b0 81 32 9f b5 55 de 6e f4 60 23 7d ff 8b e4 ba }
    $basePoint_x962P239v3 = { 04 67 68 ae 8e 18 bb 92 cf cf 00 5c 94 9a a2 c6
d9 48 53 d0 e6 60 bb f8 54 b1 c9 50 5f e9 5a 16 07 e6 89 8f 39 0c 06 bc 1d 55
2b ad 22 6f 3b 6f cf e4 8b 6e 81 84 99 af 18 e3 ed 6c f3 }
    $basePoint_x962P256v1 = { 04 6b 17 d1 f2 e1 2c 42 47 f8 bc e6 e5 63 a4 40
f2 77 03 7d 81 2d eb 33 a0 f4 a1 39 45 d8 98 c2 96 4f e3 42 e2 fe 1a 7f 9b 8e
e7 eb 4a 7c 0f 9e 16 2b ce 33 57 6b 31 5e ce cb b6 40 68 37 bf 51 f5 }

condition:
    $ecc_custom_params
```


© 2022 The SANS Institute Author retains full rights.

```
    and not any of ($basePoint*)

}
```

© 2022 The SANS Institute, Author Retains Full Rights

© 2022 The SANS Institute Author retains full rights.