Curveball+: Exploring Curveball-Like Vulnerabilities of Implicit Certificate Validation

Yajun Teng[1,2], Wei Wang[1(✉)], Jun Shao[3], Huiqing Wan[1,2], Heqing Huang[1], Yong Liu[4], and Jingqiang Lin[5]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing 100085, China
wangwei@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
[3] School of Cyber Security, Zhejiang Gongshang University, Hangzhou 310018, Zhejiang, China
[4] Qi An Xin Technology Group Inc, Beijing 100012, China
[5] School of Cyber Security, University of Science and Technology of China, Hefei 230027, Anhui, China

Abstract. The Curveball vulnerability exploits defective ECC public-key comparisons without matching domain parameters on X.509 certificates in MS Windows. Attackers can forge certificate chains that have the same public key value as a Windows-trusted certificate to establish fake HTTPS websites or sign malware binaries, which will be successfully verified without any alerts. This paper expands the Curveball attack to Elliptic-curve Qu-Vanstone implicit certificates, which are ECC-specific and have reduced certificate size and computation cost of certificate validation. We present two versions of the Curveball+ attack that target the implicit certificate validation where the verifiers are prone to the Curveball vulnerability. We discuss different types of certificate chains, implicit and hybrid, and various certificate trust list entry structures and certificate formats. We prove that verifiers that compare the final public key of implicit certificates are secure against Curveball+ version 1 attacks, but Curveball+ version 2 attacks will succeed certificates in M2M format due to the assailable standard description. Our work has preventive values for developers to avoid some of the potential implementation pitfalls.

Keywords: Curveball vulnerability · Implicit certificate · ECC

1 Introduction

The *Curveball* vulnerability was discovered and first publicized by National Security Agency (NSA) [19] in January 2020, numbered CVE-2020-0601. It is a spoofing flaw in validating X.509 certificate chains on Windows 10 and Windows

This work was supported by the National Key R&D Program of China (Award No.2020YFB1005800). Wei Wang is the corresponding author.

G. Tsudik et al. (Eds.): ESORICS 2023, LNCS 14345, pp. 212–234, 2024.
https://doi.org/10.1007/978-3-031-51476-0_11

Server 2016/2019 [20]. When inspecting the ECC public keys during matching certificate trust list (CTL) entries, the deficient MS Windows ignores the comparison of ECC domain parameters, particularly the generator. By exploiting this defect, an attacker can produce a certificate with an arbitrary subject without knowing the private key for a legitimate certificate and enable the vulnerable MS Windows to validate this certificate successfully. As a result, the vulnerability might cause severe consequences with no warning, such as letting Windows accept any malicious code with a suspect code signature or entering a fake HTTPS website with the Edge browser [25,26,28].

Elliptic-Curve Qu-Vanston scheme (ECQV) [7] implicit certificates (referred to as *impCerts*) offer performance advantages in size and validation computation, compared to conventional explicit X.509 certificates (referred to as *expCerts*). ECQV certificates use the EC-Schnorr signature technique [27] in key pair generation, ensuring integrity and authenticity without a signature field. Validation combines the public key reconstruction and the further signature verification of a signed message, improving performance by eliminating a scalar multiplication step. Despite the lack of the signature field, attackers cannot forge valid implicit certificates. Several fundamental applications attempt to use ECQV implicit certificates, such as the Internet of Vehicles [24,30], ZigBee [4], NFC [12], and TLS/DTLS [21].

ECC-based *expCerts* are vulnerable to Curveball attacks when domain parameters are improperly processed; *impCerts* are also ECC-based (ECC-specific, actually). Each certificate in the *explicit* certificate chain might employ a different ECC algorithm, where the generator should be used to confirm the signature of the lower-level certificate; in contrast, the process of public key reconstruction under the *implicit* certificate chain does not involve generators since both upper- and lower-level certificates utilize the same ECC domain parameters. In light of the investigation above, a question is raised below. *Does the deficient validation of impCerts suffer from some Curveball-like attacks?*

It is not easy to directly answer this question for the following reasons.

1. The provable security of the ECQV scheme [7] is inapplicable to answer the above question directly. Predetermined domain parameters are premised on ECQV's security model. This is unsuitable for Curveball-like attacks that involve different generators.
2. A variety of implicit certificate scenarios requires additional and intricate analysis. Concretely, the public key infrastructures (PKIs) supporting implicit certificates can be built hierarchically with implicit certificates only or in a hybrid way (i.e., with both explicit and implicit certificates).
3. Specific implicit certificate formats may also affect the success of attacks. Various formats such as X.509-compliant [9], MES [9], M2M [11], IEEE 1609.2 [14], and ETSI [10], designed for multiple applications before the disclosure of Curveball. Further analyses are necessary to answer this question.

In this paper, we re-examine the original Curveball attack to various spoofing attacks, which we call *Curveball+*. This is a much larger family of Curveball attacks against verifiers validating implicit or hybrid certificate chains (i.e.,

implicit or hybrid verifiers) with similar Curveball vulnerabilities and influences to MS Windows. Our contributions and conclusions are described as follows.

1. We propose two independent versions of Curveball+ attacks targeting positions where the implicit verifiers are likely to ignore the domain parameter comparisons: matching the CTL entry (version 1) and reconstructing the final public key (version 2). We prove that if they store and check the final public key rather than the reconstruction value of the certificate, the Curveball+ v1 attacker will fail due to the impossibility of generating its forged root impCert.
2. We also cover the cross-type Curveball+ v1 attacks against hybrid verifiers. We find that ignoring the certificate type when matching the CTL entry is sufficient, but attacks occur if the verifiers also have improper processes of domain parameters.
3. Moreover, we analyze the existing five certificate formats supporting impCerts. We conclude that MES, IEEE 1609.2, and ETSI are safe against Curveball+ attacks with their restrictive designs, while M2M, with an inherent flaw in its design, allows for successful Curveball+ v2 attacks.

2 Background

2.1 ECC Domain Parameters in X.509 Certificates

Elliptic curve domain parameters $\mathbb{E}(G)$ over $\mathbb{F}_p$ are a sextuple: (p, a, b, G, n, h), where the first triple (p, a, b) is EC parameters for point additions and scalar multiplications. The generator G, its order n and its cofactor h are mainly used in ECC algorithms, such as ECDSA [15], EC-Schnorr [27], and SM2 [3].

Domain parameters have been embedded in X.509 certificates as the parameters of a subject's public key. Two alternative formats are supported [23]: ① an object identifier (OID) to specify the pre-defined sextuple; and ② all sextuple explicitly included and encoded (i.e., support customized domain parameters).

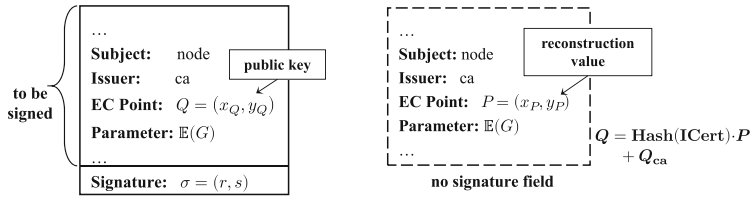
2.2 ECQV Implicit Certificates

Elliptic Curve Qu-Vanston(ECQV) scheme [9] is the most prominent impCert issuing scheme. As shown in Fig. 1, the EC Point P in an impCert is called *the reconstruction value*, and *the final public key value* $Q \neq P$ of the certificate holder needs to be reconstructed, whereas $Q = P$ for an expCert.

Issuance. Given the domain parameter $\mathbb{E}(G)$ and a collision-resistant hash function Hash, the process of a user U applying for an ECQV impCert $\mathsf{ICert}_U[P_U, \mathbb{E}(G)]$ from a CA holding a key pair (d_{CA}, Q_{CA}) can be described as follows.

1. The user selects $k_U \in [1, n)$ and sends $\{U, R_U := k_U \cdot G\}$ to the CA.

2. The CA selects another $k \in [1, n)$, computes the reconstruction value $P_U := R_U + k \cdot G$, and generates the impCert $\mathsf{ICert}_U[P_U, \mathbb{E}(G)]$.
3. The CA computes $r := e \cdot k + d_{CA} \bmod n$, where $e := \mathrm{Hash}(\mathsf{ICert}_U)$.
4. After receiving $\{\mathsf{ICert}_U[P, \mathbb{E}(G)], r\}$, the user calculates its private key $d_U := e \cdot k_U + r \bmod n$ and *reconstructs* the public key $Q_U := e \cdot P + Q_{CA}$.



(a) ECC explicit certificates (b) ECQV implicit certificates

Fig. 1. An overview of ECQV implicit certificates and ECC explicit certificates.

Validation. When verifying signed data under an impCert, the verifier first reconstructs the final public key and then uses it to verify the data signature. Compared to verifying an ECDSA signature, the reconstruction saves one scalar multiplication operation, which speeds up the certificate validation.

The generator G is *common* of two key pairs: ① the CA's key pair, i.e., $Q_{CA} = d_{CA} \cdot G$; and ② the user's key pair, i.e., $Q_U = d_U \cdot G$. Thus, the identical G is also required for the user to sign messages, denoted as $\mathrm{Sign}(m, d_U)|_{\mathbb{E}(G)}$.

Implicit Chains. Implicit certificates compose a chain when CAs cooperate hierarchically, where a CA applies for its impCert from its upper-level CA. The root CA of an implicit certificate chain works as follows [9]. It generates a random integer $k_R \in [1, n)$, computes $P_R := k_R \cdot G$ as the public key reconstruction value, and publishes its "self-signed" certificate $\mathsf{ICert}^{\mathrm{R}}[P_R, \mathbb{E}(G)]$. Its private key is $d_R = e_R \cdot k_R \bmod n$ where $e_R = \mathrm{Hash}(\mathsf{ICert}^{\mathrm{R}})$, and the public key $Q_R := e_R \cdot P_R$. Given an implicit certificate chain $\left\{\mathsf{ICert}_i, \mathsf{ICert}_{i-1}, \ldots, \mathsf{ICert}_0^{\mathrm{R}}\right\}$, the end entity i's public key is computed by $Q_i := \sum_{k=0}^{i} \mathrm{Hash}(\mathsf{ICert}_k) \cdot P_k$. Note that the generators of all impCerts in the chain need to be consistent.

3 Curveball Vulnerability on MS Windows

This section explains the Curveball vulnerability in matching certificate trust list (CTL) entries on MS Windows.

CTL is a local list of all root/intermediate CA certificates trusted by Windows. During the certificate chain validation, the verifier matches the top-level certificate with CTL entries and verifies the CAs' signatures of lower-level certificates. In MS Windows, CTL is referred to as a "certificate cache" in the memory, expediting the validation process.

3.1 Attack Overview

A Curveball attacker can exploit the Curveball vulnerability to create a *forged certificate* that matches $Q^\star$ in some CTL entries while binding another key pair $(d', Q^\star)$ without breaking any ECC signature algorithms.

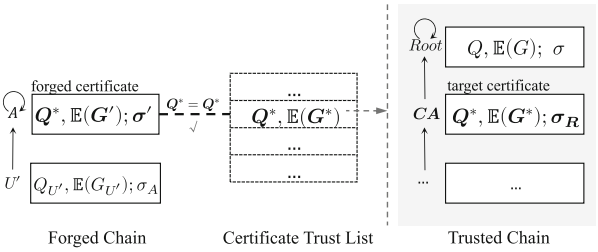


Fig. 2. Overview of the Curveball attack in explicit certificates.

The Curveball attack against a vulnerable explicit verifier is outlined in Fig. 2. The attacker begins by selecting an ECC expCert (the *target certificate*) trusted by the verifier. First, it generates a private key d' and the forged generator G' using a special key generation process. Next, it creates the forged expCert $\mathsf{ECert}^{\mathrm{R}}_A[Q^\star, \mathbb{E}(G'); \sigma']$, where the signature $\sigma' := \mathrm{Sign}([Q^\star, \mathbb{E}(G')], d')\big|_{\mathbb{E}(\mathbb{G}))}$. Then, it signs another expCert $\mathsf{ECert}_{U'}[Q_{U'}, \mathbb{E}(G_{U'}); \sigma_A]$ to bind any key pair $(d_{U'}, Q_{U'})$ of U' where $\sigma_A := \mathrm{Sign}([Q_{U'}, \mathbb{E}(G_{U'})], d')\big|_{\mathbb{E}(G')}$. Finally, it outputs a certificate chain $\{\mathsf{ECert}_{U'}, \mathsf{ECert}^{\mathrm{R}}_A\}$ with the private key $d_{U'}$. A *successful Curveball attack*[1] is that the attacker ① constructs a key pair whose public key is accepted by the verifier's validation algorithm, and ② should not require the participation of the private key corresponding to the CTL entry.

3.2 Attack Details

The key generation process is the core of the original Curveball attack. By executing Algorithm 1, the attacker can get a key pair $(d', Q^\star)$ on $\mathbb{E}(G')$, where the public key consists of *identical* (x, y)-coordinates and curves to the CTL entry $[Q^\star, \mathbb{E}(G^\star)]$ but on *different* generators $G' \neq G^\star$. Since $\mathbb{E}$ is a finite field, and the multiplicative group of EC points except $\mathcal{O}$ is cyclic, other tuples (e.g., the order and the cofactor) will *not* change. The remaining fields of the forged certificate, such as the subject and extension fields, are arbitrary.

[1] The definition is also suitable for our Curveball+ v1/v2 attacks.

Algorithm 1. Key Generation of an original Curveball Attack

Require: CTL entry $[Q^\star, \mathbb{E}(G^\star)]$
1: randomly select $d' \in [1, n)$
2: $t := {d'}^{-1} \bmod n$
3: $G' := t \cdot Q^\star$
4: **return** d', G'

When validating the received certificate chain, the vulnerable verifier will accept and use the public key value $Q_{U'}$ with its domain parameters $\mathbb{E}(G')$. $(Q_{U'}, \mathbb{E}(G'))$ are under the attacker's complete control, with the corresponding private key $d_{U'}$, but not certified by any CA in the verifier's CTL, resulting in a *successful* Curveball attack.

3.3 Vulnerability Discussions

A complete chain validation on Windows without Curveball vulnerability is described in Algorithm 2[2] where the verifier accepts and outputs the end entity's public key $(Q_i, \mathbb{E}(G_i))$ for future use. It is feasible whether the top-level certificate ECert_0 is self-signed or not.

Matching the CTL entry is described in Lines $1 \sim 3$, where MS Windows compares the fingerprint of public keys, i.e., the digest, not the complete certificate (e.g., in OpenSSL). This approach draws on the idea of the Certificate Key Matcher (CKM) for checking whether a private key matches a certificate. For example, a matcher in SSLShopper [2] compares the digests of public keys between the private key and the certificate.

However, the deficient Windows only compares the (x, y)-coordinates of two public keys. In Line 1, it returns false only if $(Q_0 \neq Q^\star)$. After matching the public key of ECert_0 with a CTL entry, the deficient verifier directly uses Q_0 and $\mathbb{E}(G_0)$ to verify other expCerts with independent generators.

Algorithm 2. Validation of an explicit certificate chain

Require: An explicit certificate chain $\{\mathsf{ECert}_i, \mathsf{ECert}_{i-1}, \ldots, \mathsf{ECert}_0^{(\mathrm{R})}\}$ where ECert_k contains $[Q_k, \mathbb{E}(G_k); \sigma_{k-1}]$
Ensure: CTL entry$[Q^\star, \mathbb{E}(G^\star)]$ of $\mathsf{ECert}^{(\mathrm{R})}$
1: **if** $Q_0 \neq Q^\star$ **or** $\mathbb{E}(G_0) \neq \mathbb{E}(G^\star)$ **then**
2: **return** not-valid
3: **end if**
4: **for** $k := 1$ **to** i **do**
5: **if** $\mathrm{Verify}(\mathsf{ECert}_k, Q_{k-1}, \sigma_{k-1})|_{\mathbb{E}(G_{k-1})} =$ **false then**
6: **return** not-valid
7: **end if**
8: **end for**
9: **return** $Q_i, \mathbb{E}(G_i)$

[2] For simplicity, the algorithm only displays one CTL entry but the full CTL list is used. In addition, several crucial checks are omitted, but they are irrelevant to our analysis and can be easily modified by a Curveball attacker.

Why Does MS Windows only Match Public Keys? McAfee [16] offers its insight: It takes into account changes in the subject field due to the company's renaming or acquisition. The CKM method matches the new certificate received with the old certificate locally, which is more convenient for offline verifiers.

Why Does Curveball Attack only Exist in ECC Certificates? The ECC public key format in X.509 certificates differs from the previous RSA format. Big integers N in RSA are stored in *the key value* because each user holds a different value; however, generators G in ECC are almost the same under a selected elliptic curve, and it exists in *the key parameters*. The CKM matching method did not consider ECC public keys, resulting in the Curveball vulnerability.

Two CTL Entry Designs. TrendMicro [28] and Qi'anXin [25] conducted details of Windows' Curveball vulnerability. The vulnerability in CryptoAPI, the built-in closed-source cryptographic library in Windows, has two main entrances. The first is the normal method `CertGetCertificateChain`, which caches the full certificate content of each CTL entry but only compares the MD5 digest of public keys. The second is `CertVerifyCertificateChainPolicy`, used to check the special validity of the certificate chain [1]. It only caches and compares the SHA256 digest of the public keys in each CTL entry. We refer to the former CTL entry design as *full-size* and the latter as *limited-size*. The subsequent analysis of implicit and hybrid chains will be based on these two CTL entry designs.

4 Curveball+ v1 Attacks Against Implicit Chains

This section extends the original Curveball attack to an impersonation against pure implicit certificate chains described in SEC4 [9].

We consider the *implicit verifiers* that match their CTL entries with the same idea as the CKM. They have the same flaw that ignores matching public-key parameters as explicit verifiers. The Curveball attacks against these implicit verifiers are referred to as *Curveball+ version 1*. Note that the implicit certificates mentioned in Sects. 4, 5, 6 allow custom domain parameters, such as the X.509-compliant format described in SEC4 [9].

4.1 Overview of Curveball+ v1 Attacks

When matching a CTL entry for an impCert, the negligence in matching the ECC domain parameters allows the attackers to deceive the verifier by creating different domain parameters, making Curveball+ v1 attacks feasible.

In the Curveball+ v1 attack, the attacker selects a target impCert $\mathsf{ICert}^{\mathrm{R}}[P^{\star}, \mathbb{E}(G^{\star})]$, generates a forged root impCert $\mathsf{ICert}_{A}^{\mathrm{R}}[P', \mathbb{E}(G')]$ and a private key d' using specific key generation algorithms, where the forged generator $G' \neq G^{\star}$. The remaining fields of the forged certificate are also arbitrary. Next, the attacker signs an impCert $\mathsf{ICert}_{U'}[P_{U'}, \mathbb{E}(G')]$ with the same forged ECC parameter $\mathbb{E}(G')$

through the ECQV procedure using d'. The corresponding private key is $d_{U'}$. Finally, the attacker signs a malicious message with $d_{U'}$, and outputs the certificate chain $\{\mathsf{ICert}_{U'}, \mathsf{ICert}_A^{\mathrm{R}}\}$.

If the attacker has a "key generation algorithm" that allows the vulnerable verifier to match the target CTL entry, the verifier will successfully verify the signature with $Q_{U'}$ and $\mathbb{E}(G')$ due to the same reconstruction, thus accepting the malicious message and the output implicit chain.

4.2 Attack Details

Next, we explain whether and how the attacker completes its key generation algorithm with the given CTL entry.

Possible v1 Attack against Implicit Verifier $\mathcal{V}_P$. If the implicit verifier stores and compares the content of the `PublicKey` field, i.e., the public-key reconstruction value P in the CTL entry, the Curveball+ v1 attack will succeed. We name this implicit verifier $\mathcal{V}_P$. Specifically, it performs Algorithm 3 to generate its forged key pair, where the forged root impCert $\mathsf{ICert}_A^{\mathrm{R}}$ is generated during the algorithm before calculating the final private key d'. The algorithm ensures equivalence of reconstruction values $P' = P^\star$ but contains different final public keys $Q' \neq Q^\star$. The user's final public key is $Q_{U'} := \mathrm{Hash}(\mathsf{ICert}_{U'}) \cdot P_{U'} + Q'$ satisfying $Q_{U'} = d_{U'} \cdot G'$.

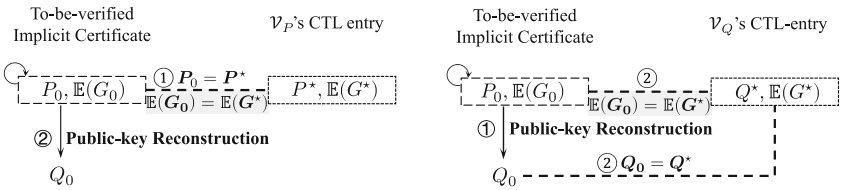
After building the certificate chain, the implicit verifier $\mathcal{V}_P$ validates whether the root certificate matches one of its local CTL entries. The matching method is shown in Fig. 3(a), where $P_0 := P' = P^\star$ and $\mathbb{E}(G_0) := \mathbb{E}(G') \neq \mathbb{E}(G^\star)$. It only compares the (x, y)-coordinates of the reconstruction value. The verifier will accept the forged root certificate $\mathsf{ICert}_A^{\mathrm{R}}$ and reconstruct the final public key $Q_0 := Q'$, thus using $(Q_{U'}, \mathbb{E}(G'))$ for subsequent signature verification. Therefore, when $\mathcal{V}_P$ is deficient in matching CTL entries, such a Curveball+ v1 attack will succeed.

Algorithm 3. Key Generation of Curveball+ v1 Attack against the implicit verifier $\mathcal{V}_P$

Require: CTL entry $[P^\star, \mathbb{E}(G^\star)]$ of $\mathsf{ICert}^{\mathrm{R}}$
1: randomly select $k \in [1, n)$
2: $t := (k)^{-1} \bmod n$
3: $G' := t \cdot P^\star$
4: generate certificate $\mathsf{ICert}_A^{\mathrm{R}}[P^\star, \mathbb{E}(G')]$
5: $e' := \mathrm{Hash}(\mathsf{ICert}_A^{\mathrm{R}})$
6: $d' := e' \cdot k \bmod n$
7: $Q' := e \cdot P^\star$
8: **return** $(d', Q'), \mathsf{ICert}_A^{\mathrm{R}}$

Impossible v1 Attack against Implicit Verifier $\mathcal{V}_Q$. Assuming that the verifier stores and compares the final public keys for subsequent use, we refer

to this verifier as $\mathcal{V}_Q$. It reconstructs the final public key Q_0 before comparing it with its CTL entry, as shown in Fig. 3(b). The deficient $\mathcal{V}_Q$ ignores matching the domain parameters $\mathbb{E}(G_0)$ and $\mathbb{E}(G^\star)$, the same as the vulnerable $\mathcal{V}_P$. To induce the verifier to trigger the vulnerability, the attacker should make the final public key Q' identical to the target one $Q^\star$ in $\mathcal{V}_Q$'s CTL entry $[Q^\star, \mathbb{E}(G^\star)]$. By executing Algorithm 1 with the CTL entry, the attacker can obtain its forged key pair $(d', Q^\star)$ on $\mathbb{E}(G')$ where $G' := (d')^{-1} \cdot Q^\star$. However, the difficulty occurs in the following stage, when the attacker must calculate the reconstruction value P' before generating its forged certificate $\mathsf{ICert}_A^{\mathrm{R}}[P', \mathbb{E}(G')]$, satisfying $Q' := \mathrm{Hash}(\mathsf{ICert}_A^{\mathrm{R}}) \cdot P' = Q^\star$.



(a) $\mathcal{V}_P$ compares reconstruction values. (b) $\mathcal{V}_Q$ reconstructs before comparison.

Fig. 3. CTL matching methods of implicit verifiers.

We define the problem of obtaining a suitable public key reconstruction value P' from a given public key $Q^\star$ as a *From-Q-to-P* (Q2P) problem, which will be proved to be hard to solve.

- If the attacker chooses a value of P', it cannot guarantee that the hash value satisfies the equation above, which is equivalent to an ECDLP problem.
- If the attacker observes a digest of the certificate, it can calculate a P but not the preimage of the hash value due to the collision resistance.

Rigorous proofs can be found in Appendix 1. Thus, the attacker cannot construct its forged impCert, and the vulnerable verifier $\mathcal{V}_Q$ with Curveball+ v1 flaw is *secure*.

4.3 Discussions

Implicit certificates can evade Curveball+ v1 attacks if the deficient implicit verifier compares the final public key value. This conclusion differs from the traditional Curveball attack described in Sect. 3 due to the unique public-key reconstruction of the impCert. Next, we discuss the two types of implicit verifiers $\mathcal{V}_Q$ and $\mathcal{V}_P$ described above.

Applicability of Two Implicit Verifiers. Since Windows Crypto API of certificate validation and other open-source implementations do not support impCerts, the discussion is based on the existing CTL comparison methods.

$\mathcal{V}_P$ matches the certificate content by comparing the EC point and its domain parameters, similar to MS Windows. After the successful matching, the verifier accepts the received top impCert, and follows the public-key reconstruction without any involvement of the CTL entry. The matching method described in Fig. 3(a) is similar to that in expCerts, and works for both *full-size* and *limited-size* CTL entry designs.

$\mathcal{V}_Q$ ensures a trusted final public key, which does not exist in the certificate content. The verifier first reconstructs the final public key of the top impCert of the chain, then matches the CTL entries, as shown in Fig. 3(b). It is only suitable for *limited-size* CTL entry design.

5 Curveball+ v1 Hybrid Attacks

This section covers the *hybrid verifiers* that support general hybrid chains where implicit certificates coexist with explicit ones.

Hybrid certificate signings are feasible in principle for two reasons: ① It is common that an expCert is the *superior* to an impCert. Such as SCMS, the vehicles' pseudonym certificates are usually implicit, but their issuers hold non-pseudonym explicit ones. ② It is suitable that an expCert is the *inferior* to an impCert. The expCert can be "a signed message" with the impCert. We present and analyze the Curveball+ v1 attacks against hybrid verifiers in general cases where the explicit and implicit certificates can be issued from one another.

5.1 Attack Overview

Curveball+ v1 hybrid attacks extend the Curveball+ v1 attacks described above to arbitrary types of target, forged, and user certificates $\{\mathsf{Cert}^{\mathrm{R}}, \mathsf{Cert}^{\mathrm{R}}_A, \mathsf{Cert}_{U'}\}$. The attacker's choice depends on whether the verifier supports the corresponding chain validation. A successful Curveball+ v1 hybrid attack occurs if the vulnerable verifier accepts $\mathsf{Cert}^{\mathrm{R}}_A$, uses the public key $Q_{U'}$ with the domain parameter $\mathbb{E}(G')$, and finally accepts any malicious message signed by $d_{U'}$.

Table 1. Four possible solutions for CTL entry of hybrid verifiers.

CTL entry	Verifier	Characteristics of the verifier
$[P^\star, \mathbb{E}(G^\star), \mathrm{Tag}]$	$\mathcal{V}_{T,P}$	store and match P with the Certificate type
$[Q^\star, \mathbb{E}(G^\star), \mathrm{Tag}]$	$\mathcal{V}_{T,Q}$	store and match Q with the Certificate type
$[P^\star, \mathbb{E}(G^\star)]$	$\mathcal{V}_{N,P}$	store and match P without the certificate type
$[Q^\star, \mathbb{E}(G^\star)]$	$\mathcal{V}_{N,Q}$	Store and match Q without the certificate type

Four possible hybrid solutions for CTL entry are described in Table 1. The first two verifiers $\mathcal{V}_{T,*}$ contain a *tag* to indicate the certificate type. The attacker's forged certificate will have the same type as the target, which is already described in Sects. 3 and 4, except for the arbitrary user certificate type. The remaining two, $\mathcal{V}_{N,*}$, mainly focus on the cross-type attacks where hybrid verifiers with Curveball vulnerability does not have the *tag* in their CTL entries. Note that the user certificate type is unrelated to the deficient CTL comparison, so we will detail specific key generations and forged certificates of cross-type Curveball+ attacks against verifiers $\mathcal{V}_{N,*}$.

5.2 Attack Details

First, we consider the vulnerable verifier $\mathcal{V}_{N,P}$. Attackers can launch both cross-type Curveball+ v1 attacks successfully.

- After selecting the target expCert $\mathsf{ECert}^{\mathrm{R}}[P^\star, \mathbb{E}(G^\star); \sigma]$, the attacker runs Algorithm 3 to obtain its generator $G' := k^{-1} \cdot P^\star$ and private key $d' := \mathrm{Hash}(\mathsf{ICert}^{\mathrm{R}}_A) \cdot k \bmod n$. The forged root impCert $\mathsf{ICert}^{\mathrm{R}}_A[P', \mathbb{E}(G')]$ is already constructed during this algorithm.
- After selecting the target impCert $\mathsf{ICert}^{\mathrm{R}}[P^\star, \mathbb{E}(G^\star)]$, the attacker runs Algorithm 1 with the given $P^\star$ to obtain its random private key d' and generator $G' := (d')^{-1} \cdot P^\star$. The public key of the forged certificate satisfies $Q' = P^\star$. Therefore, the attacker's forged self-signed expCert is $\mathsf{ECert}^{\mathrm{R}}_A[Q', \mathbb{E}(G'); \sigma']$, where the signature σ' is generated by the forged domain parameter $\mathbb{E}(G')$.

Next, we consider the vulnerable verifier $\mathcal{V}_{N,Q}$. In this case, the attacker can launch a Curveball+ v1 cross-type attack against a target implicit certificate.

- After selecting the target impCert $\mathsf{ICert}^{\mathrm{R}}[P^\star, \mathbb{E}(G^\star)]$, the attacker first calculates the target's final public key $Q^\star := \mathrm{Hash}(\mathsf{ICert}^{\mathrm{R}}) \cdot P^\star$ and runs Algorithm 1 with the input $Q^\star$ to fetch its private key d' and generator $G' := (d')^{-1} \cdot Q^\star$. The public key of the forged expCert is $Q' = Q^\star$.
- After selecting the target expCert $\mathsf{ECert}^{\mathrm{R}}[P^\star, \mathbb{E}(G^\star); \sigma]$, the attacker runs Algorithm 1 then it needs to calculate P' to generate the forged impCert $\mathsf{ICert}_A[P', \mathbb{E}(G')]$ such that $Q' := \mathrm{Hash}(\mathsf{ICert}^{\mathrm{R}}_A) \cdot P' = Q^\star$. This is the same as the complex *Q2P* problem, so the attacker cannot continue this attack.

5.3 Discussions

Summary of Curveball+ v1 Hybrid Attacks. Table 2 shows whether Curveball+ v1 attacks are successful when facing different hybrid verifiers with deficient CTL entry matching. Line $\mathrm{ATK}_{E,E}$ indicates the conclusion in Sect. 3, whereas Line $\mathrm{ATK}_{I,I}$ denotes that in Sect. 4. The table shows that the weak comparison in matching CTL entries significantly impacts $\mathcal{V}_{N,P}$, i.e., the attacker can launch attacks with arbitrary modes; $\mathcal{V}_{T,Q}$ is unaffected, and only $\mathrm{ATK}_{E,E}$ is viable.

Applicability of Hybrid Verifiers. The unmatching of the certificate types is possible for hybrid verifiers since MS Windows does not check the self-signed features and signatures. The *full-size* CTL entry format contains the whole certificate, which belongs to the verifier $\mathcal{V}_{T,P}$. If the verifier does not match the certificate type, it will become $\mathcal{V}_{N,P}$. In contrast, the *limited-size* CTL entry format can be adapted to any of the four hybrid verifiers.

Security of Hybrid Verifiers. It is clear that all four verifiers have no additional flaws if they do not have the Curveball vulnerability. On the surface, $\mathcal{V}_{N,P}$ sounds irrational: the untrusted certificate may be matched to a trusted certificate of another type improperly due to the same EC point value. But in reality, it is *secure* if it does not have any Curveball flaws. Security proofs in Appendix 2 indicate the truth: a third-party attacker cannot convert the target certificate into another type that the verifier can mistakenly accept.

Table 2. Result of Curveball+ v1 hybrid attacks against vulnerable verifiers.

v1 Attack Mode	Certificate		Hybrid Verifier			
	Target Cert	Forged Cert	$\mathcal{V}_{N,P}$	$\mathcal{V}_{N,Q}$	$\mathcal{V}_{T,P}$	$\mathcal{V}_{T,Q}$
$\text{ATK}_{E,E}$	ECert	ECert	✓	✓	✓	✓
$\text{ATK}_{E,I}$	ECert	ICert	✓	×	×	×
$\text{ATK}_{I,E}$	ICert	ECert	✓	✓	×	×
$\text{ATK}_{I,I}$	ICert	ICert	✓	×	✓	×

The last four columns show whether the attacker can complete the Curveball+ v1 attack using a specific attack method under a specific hybrid verifier. "✓" indicates a successful attack, while "×" represents not.

6 Curveball+ v2 Attacks

This section discusses *Curveball+ version 2* attacks, which target implicit verifiers or hybrid verifiers. The attack exploits a flaw where the verifier does not match the public-key parameters of an implicit certificate with those in its superior. This flaw is exclusive to validating implicit certificates, irrelevant to the CKM matching method, and independent of the original Curveball vulnerability, but the two versions of attacks both neglect to align ECC domain parameters during certificate validations.

6.1 Attack Overview

Similar to a v1 attack, the v2 attack aims at forging a key pair with the same key value but different domain parameters. With a target root certificate $\mathsf{Cert}^{\mathrm{R}}$ containing $\mathbb{E}(G^{\star})$, the main difference between the two attacks is that the v2

attacker directly uses the forged key pair (d', Q') with $\mathbb{E}(G')$ for signing an impCert $\mathsf{ICert}_{U'}$, without generating its forged certificate Cert_A. The issuer of the user impCert $\mathsf{ICert}_{U'}$ should be the same as the subject of the target certificate $\mathsf{Cert}^{\mathrm{R}}$, and the output chain is $\{\mathsf{ICert}_{U'}, \mathsf{Cert}^{\mathrm{R}}\}$, which holds $G' \neq G^{\star}$. However, the vulnerable verifier omits the generator comparisons when reconstructing the public key of impCerts, thus directly using G' and successfully accepting the output signature and the "fake" certificate chain.

6.2 Attack Details

Unlike version 1, the version 2 attacker aims to make its (final) public-key value the same as that of the target certificate, either for vulnerable $\mathcal{V}_{*,Q}$ or $\mathcal{V}_{*,P}$, as described in Fig. 4. In detail, it can obtain its final private key d' and the domain parameters $\mathbb{E}(G')$ by executing Algorithm 1 with the input $[Q^{\star}, \mathbb{E}(G^{\star})]$.

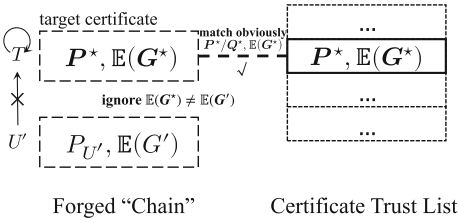


Fig. 4. Overview of a Curveball+ v2 attack targeting both $\mathcal{V}_{*,P}$ and $\mathcal{V}_{*,Q}$.

Table 3 shows the result of the Curveball+ v2 attacks targeting the hybrid verifiers. Whether the verifier is $\mathcal{V}_{*,Q}$ or $\mathcal{V}_{*,P}$ does not affect the key generation of the Curveball+ v2 attacker. Thus, all cells in the table will become "✓".

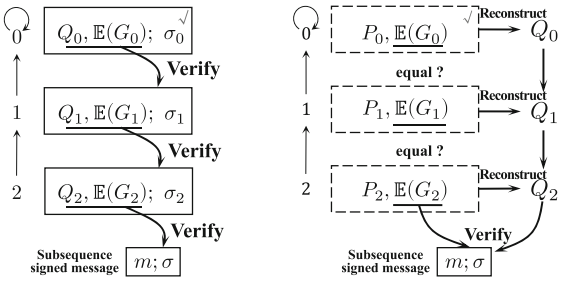
Table 3. Result of Curveball+ v2 attacks against vulnerable hybrid verifiers.

v2 Attack Mode	Target Certificate	Hybrid Verifier			
		$\mathcal{V}_{N,P}$	$\mathcal{V}_{N,Q}$	$\mathcal{V}_{T,P}$	$\mathcal{V}_{T,Q}$
$\mathrm{ATK}_{E,*}$	ECert	✓	✓	✓	✓
$\mathrm{ATK}_{I,*}$	ICert	✓	✓	✓	✓

6.3 Discussions

Now we discuss the possible vulnerability in the public-key reconstruction procedure that can lead to Curveball+ v2 Attacks.

Usage of Generators. First, v2 attacks only exist in validating impCerts. Figure 5 illustrates the different usage of generators in three-tier chains. In the implicit case, only G_2 of the end-entity is directly used for future actions, and equivalent generators indicate G_2's confidence; while in explicit cases, they are directly used for certificate signature verifications without equalities. Implicit/hybrid verifiers vulnerable in public-key reconstruction cannot establish the connection between G_2 and dependable $G_0 := G^\star$.



(a) an explicit verifier (b) an implicit verifier

Fig. 5. Different generators' usage between verifiers with a three-tier chain.

SEC4 Standard. However, the standard does not emphasize the matching of generators between an impCert and its CA. While the standard specifies that the reconstructed user's public key should be defined over the same elliptic curve as CA's public key (the third subheading of Appendix B of [9]), "the same curve" only pertains to the EC parameters (p, a, b) but not the generator G. The Curveball+ v2 attack shows that the ECC domain parameters sextuple (p, a, b, G, n, h) of the two certificates should ultimately be compared.

```
static int internal_verify(X509_STORE_CTX *ctx) {
    ...
    while (n >= 0) { // n: current chain's length
        EVP_PKEY *pkey = X509_get0_pubkey(xi); // xi: issuer's certificate
        X509_verify(xs, pkey);  // xs: subject's certificate
        ... // update n, xi, xs
    }
    return 1;
}
```

Fig. 6. The critical code for validating an X.509 certificate chain in OpenSSL 1.1.1l.

OpenSSL Interfaces. Furthermore, we find that domain parameter comparisons in public-key reconstruction can be easily overlooked. OpenSSL 1.1.1l provides a `while` loop in Fig. 6 where the verifier extracts the public key with its domain parameters of a certificate from top to bottom into an `EVP_PKEY`

object. Extending it to an implicit case, the verifier will replace `X509_verify` with the public-key reconstruction procedure. However, the point addition function `EC_POINT_add`, the last step of the public-key reconstruction, does not match the domain parameters of the two input `EC_POINT` objects. Therefore, developers must be mindful of the generator comparisons, which are not emphasized in the SEC4 standard, to prevent Curveball+ v2 attacks.

7 Format Factors Affecting Curveball+ Attacks

Previous sections analyzed two Curveball+ attacks by discussing implicit/hybrid verifier procedures. Additionally, the success of Curveball+ attacks requires certificate formats supporting custom ECC domain parameters. This section will discuss the impact of different certificate formats on our Curveball+ attacks.

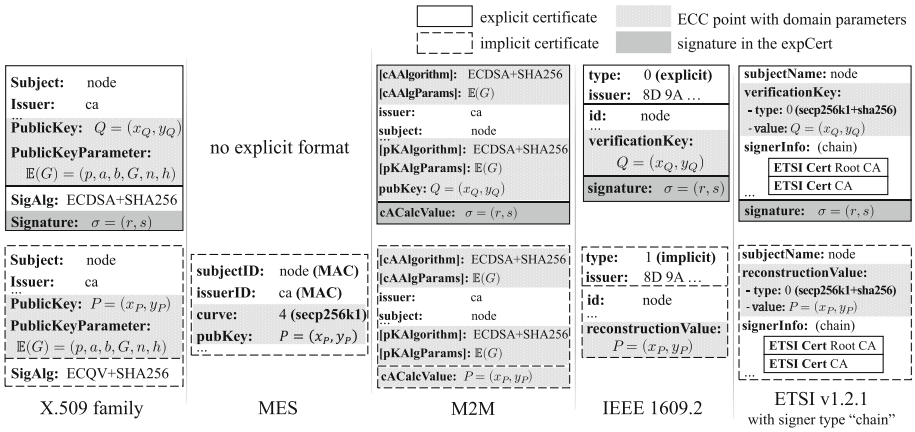


Fig. 7. Key characteristics of the five certificate formats into a simplified form.

Five certificate formats supporting implicit type are identified with various features: X.509 family [9], MES [9], M2M [11], IEEE 1609.2 [14], and ETSI v1.2 [10]. Figure 7 illustrates the intuitive differences among the total five certificate formats. Next, we sort these characteristics that are present in at least one certificate type, which may affect our Curveball+ attacks.

7.1 Basic Assumption

The basic assumption of the two versions of Curveball+ attacks is that the certificate format supports customized ECC domain parameters.

Limited Domain Parameters. MES, IEEE 1609.2, and ETSI have this restriction. They offer an enumerated number for predefined curves. MES supports all curves described in SEC2 [5], but the other two formats only contain "secp256k1". All predefined curves described in SEC2 are sextuple, containing generators. Therefore, a Curveball+ attacker cannot generate its forged/user certificate in these formats with custom domain parameters. The M2M format, based on X.509 certificate format, does not detail the public-key parameters [11]. It can be considered to support custom domain parameters.

7.2 Features Mitigating Attacks

Three other features may mitigate the success of Curveball+ attacks.

Support Implicit Certificates Only. MES has the feature. If the verifier only accepts this format, the attacker can only consider $\text{ATK}_{I,I}$.

Restrict Implicit Certificates Level. IEEE 1609.2 format has this restriction [14] due to the IoV application, where only end-entity pseudonym certificates will be implicit. The restriction also avoids the existence of a root impCert, so all Curveball+ attacks will be degraded to $\text{ATK}_{E,E}$.

Represent the Issuer by the Digest. IEEE 1609.2 and ETSI adopt the last 32-bit digest of the superior certificate as the inferior's issuer (where root certificates mark the issuer with its hash algorithm). The digest can be used directly for the *limited-size* CTL entry: [Hash(Cert)], an identical digest indicating the integrity of the certificate, including domain parameters. If the CTL entry is designed accordingly, the attacker cannot generate its forged certificate containing its custom domain parameters with the same identical digest, thus all Curveball+ v1 attacks on vulnerable CTL matching will be fruitless.

7.3 Features Magnifying Attacks

The following feature may allow for a successful Curveball+ v2 attack even if the implicit/hybrid verifier has no Curveball vulnerabilities.

Allow Optional Public-Key Parameters. Both X.509 and M2M formats allow for parameters omission, but with different methods of obtaining them. The X.509-compliant [9] allows for retrieving from superior CA certificates (until a self-signed certificate), while the M2M format has no restrictions. It is possible to extract an M2M certificate's public key parameters from its lower-level certificate's `cAAlgParams` field. Section 6 in the IETF draft [11] sets the rule for omitting algorithm fields. It specifies that `cAAlgorithm` can only be omitted when `pKAlgorithm` of a superior certificate *fully* specifies the algorithm and its domain parameters. The description allows for the situation above if the CA

certificate contains `pKAlgorithm` field but omits `pKAlgParams` field, since the `xxAlgorithm` field and the `xxAlgParams` field are optional independently.

This method can expose the M2M certificate verifier to a severe Curveball+ v2 attack without Curveball vulnerability. Targeting an M2M root impCert with no public key parameters, the attacker can launch a Curveball+ v2 attack using a user impCert $\mathsf{ICert}_{U'}$ with a different custom generator. The verifier complying with the IETF draft will ultimately extract the custom domain parameters and apply them to further validation directly.

In principle, this design of M2M is problematic. However, the vulnerability has limited impact due to two factors: ① M2M is only a draft that has not been updated since March 2015 [11], and ② there is no widely-used open-source M2M implementation, including certificate validation. We have emailed the original author and are still waiting for a response.

7.4 Features Affecting Attackers' Calculation

The other two features do not affect the success of the attack, but they influence the calculation of the Curveball+ attacker.

Calculate the Digests Uniquely. IEEE 1609.2 has this specialty. It replaces the digests $\mathrm{Hash}(\mathsf{ICert}_U)$ and $\mathrm{Hash}(m)$ with $\mathrm{Hash}(\mathrm{Hash}(\mathsf{ICert}_U)||\,\mathrm{Hash}(\mathsf{ICert}_{CA}))$ and $\mathrm{Hash}(\mathrm{Hash}(m)||\,\mathrm{Hash}(\mathsf{ICert}_U))$, respectively. This means that the signer's certificate participates in the computation of the digest.

Contain CA Certificates Inside the Body. ETSI format has this feature, which puts the complete chain into the user certificate as the `signerInfo` field.

The attacker needs to customize its Curveball+ attack to accommodate these features. It uses $\mathrm{Hash}(\mathsf{ICert}^{\mathrm{R}})$ on Curveball+ v2 attacks and $\mathrm{Hash}(\mathsf{ICert}^{\mathrm{R}}_A)$ on v1 attacks for the consistent digests in ECQV public-key reconstruction.

7.5 Summary

The results generalized by each certificate format are shown in Table 4. MES, IEEE 1609.2, and ETSI v1.2 do not lead to Curveball+ attacks because they do not support customized curve parameters. Verifiers with X.509-family and M2M format may have Curveball vulnerabilities, and they need to be careful with domain parameter comparisons. Additionally, M2M has a defect that the attacker may launch a successful Curveball+ v2 attack against an invulnerable verifier.

8 Related Work

Research of Certificate Validation. Several studies such as Frankencerts [8] generated mutant certificates automatically to test for defects in libraries and

browsers. These tests revealed many flaws in field checking of validating a certificate. In addition, research [13] focused on API usability and documentation. Their testing libraries cover multiple platforms, devices, and program languages, but none currently support implicit certificates.

ECQV's Analysis. SEC4's appendix [9] discusses commentaries on implicit certificates, including the inability to sign implicit cross-certificates and the potential use of Wagner's tree algorithm to solve the implicit chain's combined equation. Daniel et al. [6] found that the implicit user certificate cannot be the signing message when combining the ECQV algorithm with ECDSA. These studies did not involve validations of domain parameters of implicit certificates.

Table 4. Result of Curveball+ attack for different formats.

Features of Certificate Formats	Certificate Formats				
	X.509	MES	M2M	1609.2	ETSI
Limited domain parameters		×		×	×
Support implicit certificates only		−			
Restrict implicit certificates level				−	
Represent the issuer by the digest				$\times^{d}$	$\times^{d}$
Allow optional public-key parameters	O		+		
Calculate the digests uniquely				O	
Contain CA certificates inside the body					O
Curveball+ Attack Result	✓	×	✓	×	×

"+", "×" or "−" mean that the certificate format supports features that can *magnify*, *fix* or *mitigate* the Curveball attacks. "^{d}" indicates storing the certificate digest in CTL entries. "✓" or "×" denote that the weak verifier supporting this format is *vulnerable* or *secure* with Curveball+ attacks.

Studies Using Similar Analysis Thought. Nikos et al. [17] described a cross-protocol attack in TLS 1.2 based on the Wagner and Scheiner attack [29] in SSL 3.0. Similarly, this attack assumes support for custom domain parameters and is analyzed as a simulation. However, the cross-protocol attack is probabilistic and depends on the vulnerable protocol; whereas our Curveball+ attacks are conditional with vulnerable verifiers.

9 Conclusion

This paper proposes two versions of possible Curveball+ attacks against ECQV implicit certificates based on the origin Curveball attack against X.509 ECC certificates. Version 1 is for implicit and hybrid verifiers with a similar vulnerability to Windows in matching CTL entries. However, it is secure if the deficient verifiers compare the final public keys rather than the reconstruction

value. For hybrid verifiers, not comparing certificate types is not a flaw, but additional Curveball vulnerabilities enable easy launch of v1 attacks. Version 2 is for verifiers that easily ignore matching domain parameters during the public-key reconstructions of implicit certificates. Both v1/v2 attacks have the same consequences as the original Curveball attack, as they focus on all possible domain parameter comparisons during the implicit certificate validation process. We simulated all successful Curveball+ attacks presented in this paper[3]. In addition, MES, IEEE 1609.2, and ETSI avoid the possibility of attacks by limiting domain parameters, but the improper M2M design leads to v2 attacks even the verifiers without Curveball vulnerability. The latest RFC 8902 [18] catalyzes the development of implicit certificates. However, several existing drafts or standards for implicit certificate formats need improvements to be more developer-friendly in validating implicit certificates. We hope that the standard-setter of implicit certificates will provide clear procedures and designs, so that software developers will understand the design principle of implicit certificate crypto suites better and avoid some of the possible implementation traps in the future.

Appendix 1 Proofs of the Q2P Problem

The format $[m, P]$ represents the certificate $\mathsf{ICert}[P, \mathbb{E}(G)]$ in the security model, where m represents the other information of ICert except the EC Point P, renamed as the message. The rigorous definition of *Q2P* is described as follows.

Definition 1. *Given an Elliptic Curve* $\mathbb{E}$*, a hash function* Hash *and an EC Point* Q*, Q2P problem asks for a message* m *and another EC Point* P *such that* $Q = \mathrm{Hash}(m, P) \cdot P$*.*

In the random oracle model, we define the game for an adversary $\mathcal{A}$ to solve *Q2P* problem as $\mathrm{Game}_{\mathcal{A}}^{\mathrm{Q2P}}(\lambda, \mathbb{E})$[4] with a hash oracle $\mathcal{O}_{\mathcal{A}}^{\mathrm{Hash}}$. Note that the attacker in reality limits the (x, y)-coordinates of G' in the output m (Sect. 4), more complex than Definition 1 with arbitrary m.

Lemma 1. *In the random oracle model, the EC-Schnorr family of signature schemes in* $\mathbb{E}(G)$ *is secure if the ECDLP problem in* $\mathbb{E}(G)$ *is intractable.*

The variant of the *Schnorr* signature for a message m with the private key b can be expressed as $\sigma := (R, s)$ where the EC Point $R := k \cdot G$ with random secret k, and the integer $s := b + k \cdot \mathrm{Hash}(m, R)$. To verify the signature, one checks that $s \cdot G = \mathrm{Hash}(m, R) \cdot R + B$ with the public key B.

Pointcheval and Stern [22] have proved Lemma 1 by constructing a reduction from ECDLP to the variant EC-Schnorr Signatures with the "forking lemma".

Theorem 1. *In the random oracle model, Q2P problem in* $\mathbb{E}$ *is difficult if the Schnorr Signature Scheme in* $\mathbb{E}(G)$ *is secure.*

[3] See https://github.com/tyj956413282/curveball-plus.git for source code.

[4] λ represents the bit-number of $\#\mathbb{E}$ (the number of all EC points in $\mathbb{E}$).

Proof. We just reveal the following experience: assuming that there exists a successful adversary $\mathcal{A}$ solving the *Q2P* problem, construct a polynomial algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine to forgery the EC-Schnorr signature with nonnegligible probability. The game $\text{Game}_{\mathcal{B}}^{\text{Schnorr}}$ runs as follows:

1. After receiving the public key B, randomly select an integer $s \in [1, n)$ as a part of the output signature and calculate the final public key $Q := s \cdot G - B$;
2. To obtain the message and another part of the signature, run $\text{Game}_{\mathcal{A}}^{\text{Q2P}}(\lambda, \mathbb{E})$ with $\mathcal{O}_{\mathcal{B}}^{\text{Hash}}$;
3. If $\mathcal{A}$ wins with output (m', P'), construct and output the message with the forged signature $\big(m', \sigma' := (P', s)\big)$; otherwise, terminate $\perp$.

The following two factors allow $\mathcal{B}$ to pass the game, which proves the correctness.

1. **New message:** m' is suitable since $\mathcal{B}$ did not make any signature query.
2. **Signature verification:** the verification with signature σ' will be passed due to $\text{Hash}(m', P') \cdot P' + B = Q + B = (s \cdot G - B) + B = s \cdot G$.

If $\mathcal{A}$ runs in polynomial time and succeeds with nonnegligible probability, so will $\mathcal{B}$. But by hypothesis, no such $\mathcal{B}$ can make a forged variant EC-Schnorr signature in $\mathbb{E}(G)$. Therefore, no adversary $\mathcal{A}$ exists in the random oracle model, and the proof of this theorem is complete.

Combing Lemma 1 and Theorem 1, we can get that the Q2P problem is based on the ECDLP problem.

Appendix 2 Rationality of the Hybrid Verifier $\mathcal{V}_{N,P}$

The *rationality* of a hybrid verifier $\mathcal{V}_{N,P}$ is that any certificate holder, except a self-signed holder, cannot change the certificate type so that the verifier will accept it. That is, transforming an explicit certificate into implicit (*E2I*), and transforming an implicit certificate to explicit (*I2E*). The rationality of $\mathcal{V}_{N,P}$ is based on the ECDLP assumption with two additional oracles: an ECDSA signature oracle $\mathcal{O}^{\text{Sign}}$ and an ECQV certificate oracle $\mathcal{O}^{\text{ECQV}}$. Both ECDSA and ECQV algorithm are also based on the ECDLP assumption [7,15], thus our ECDLP attacker have the ability to ask $\mathcal{O}^{\text{Sign}}$ and $\mathcal{O}^{\text{ECQV}}$. To simplify our proofs, we use the explicit certificate as an example.

Transform Explicit Certificates. Assume that a trusted certificate chain $\{\mathsf{ECert}_1, \mathsf{ECert}_0^{\text{R}}\}$ is stored in the verifier $\mathcal{V}_{N,P}$. We define $\text{Game}_{\mathcal{A},\mathcal{V}_{N,P}}^{E2I}(\mathsf{ECert}, n)$ as follows: After receiving a certificate chain $\big\{\mathsf{ECert}_1[Q_1, \mathbb{E}(G_1); \sigma_0], \mathsf{ECert}_0^{\text{R}}[Q_0, \mathbb{E}(G_0); \sigma]\big\}$ with a private key d_1 where $Q_1 = d_1 \cdot G_1$, output a forged nonroot implicit certificate with a private key $(d', \mathsf{ICert}_{\mathcal{A}}[P', \mathbb{E}(G_1)])$ so that the final public key satisfies $Q' := \text{Hash}(\mathsf{ICert}_{\mathcal{A}}) \cdot P' + Q_0 = d' \cdot G_1$ and $P' = Q_1$.

Theorem 2. *In the random oracle model,* I2E *does not exist in normal* $\mathcal{V}_{N,Q}$ *if the ECDLP problem is hard to solve.*

Proof. We design $\mathrm{Game}_{\mathcal{B}}^{\mathrm{ECDLP}}(n, \mathbb{E}(G))$ using $\mathrm{Game}_{\mathcal{A},\mathcal{V}_{N,P}}^{E2I}(\mathsf{ECert}, n)$ as follows:

1. After receiving the public key B, randomly select $d_1 \in [1, n)$ and generate the two certificates $\mathsf{ECert}_0^{\mathrm{R}}[B, \mathbb{E}(G); \sigma]$, $\mathsf{ECert}_1[Q_1, \mathbb{E}(G); \sigma_0]$ where σ and σ_0 are obtained by asking the signature oracle $\mathcal{O}_{\mathcal{B}}^{\mathrm{Sign}}$ and $Q_1 := d_1 \cdot G$. Send $\left(\{\mathsf{ECert}_1, \mathsf{ECert}_0^{\mathrm{R}}\}, d_1\right)$ to $\mathcal{A}$.
2. Judge $\mathcal{A}$'s answer when $\mathcal{A}$ outputs as $(d', \mathsf{ICert}_{\mathcal{A}})$.
3. If $\mathcal{A}$ wins, calculate $d_0 = d' - \mathrm{Hash}(\mathsf{ICert}_{\mathcal{A}}) \cdot d_1$, and output $b := d_0$.

We state that $B = b \cdot G$ for correctness, under the premise of $Q_1 = P'$ in $\mathcal{V}_{N,P}$. We have $B = Q' - \mathrm{Hash}(\mathsf{ICert}_{\mathcal{A}}) \cdot P' = \left(\mathrm{Hash}(\mathsf{ICert}_{\mathcal{A}}) \cdot d_1 + b\right) \cdot G - \mathrm{Hash}(\mathsf{ICert}_1) \cdot P' = b \cdot G$. If $\mathcal{A}$ successfully constructs the eligible nonroot implicit certificate, $\mathcal{B}$ is also successful in solving the ECDLP problem, proving the theorem.

Transform Implicit Certificates. Assume that a trusted certificate chain $\left\{\mathsf{ICert}_1\ [P_1, \mathbb{E}(G)], \mathsf{ECert}_0^{\mathrm{R}}\ [B, \mathbb{E}(G); \sigma]\right\}$ is stored in the verifier $\mathcal{V}_{N,P}$. We define $\mathrm{Game}_{\mathcal{A},\mathcal{V}_{N,P}}^{I2E}(\mathsf{ICert}, n)$ as follows: After receiving $\{\mathsf{ICert}_1, \mathsf{ECert}_0^{\mathrm{R}}\}$ and (d_1, k') where $Q_1 := \mathrm{Hash}(\mathsf{ICert}_1) \cdot P_1 + Q_0 = d_1 \cdot G$, ($k'$ is defined in the ECQV procedure for ICert_1), output a forged nonroot certificate with a private key $\left(d', \mathsf{ECert}_{\mathcal{A}}[Q', \mathbb{E}(G); \sigma_0]\right)$ so that $Q' := d' \cdot G = P_1$.

Theorem 3. *In the random oracle model,* I2E *does not exist in normal* $\mathcal{V}_{N,P}$ *if the ECDLP in* $\mathbb{E}(G)$ *is hard to solve.*

Proof. We design $\mathrm{Game}_{\mathcal{B}}^{\mathrm{ECDLP}}(n, \mathbb{E}(G))$ using $\mathrm{Game}_{\mathcal{A},\mathcal{V}_{N,P}}^{I2E}(\mathsf{ECert}, n)$ as follows:

1. After receiving the public key B, generate $\mathsf{ECert}_0^{\mathrm{R}}\ [B, \mathbb{E}(G); \sigma]$ asking an ECDSA signature oracle $\mathcal{O}_{\mathcal{B}}^{\mathrm{Sign}}$ for σ.
2. Call ECQV procedure to generate $\mathsf{ICert}_1[R, \mathbb{E}(G)]$ whose CA is $\mathsf{ECert}_0^{\mathrm{R}}$, with a median k_1 and the private key d_1. Noted that the public and private key reconstruction values are (R, s) by a query of Schnorr signature Oracle $\mathcal{O}_{\mathcal{B}}^{\mathrm{Sign}}$ where m is defined in Appendix 1.
3. Send $\left(\{\mathsf{ICert}_1, \mathsf{ECert}_0^{\mathrm{R}}\}, d_1\right)$ to $\mathcal{A}$.
4. Judge $\mathcal{A}$'s answer when $\mathcal{A}$ outputs as $(d', \mathsf{ECert}_{\mathcal{A}})$.
5. If $\mathcal{A}$ wins, calculate $\mathsf{ECert}_0^{\mathrm{R}}$'s private key $d_0 := d_1 - \mathrm{Hash}(\mathsf{ICert}_1) \cdot d'$, and output $b := d_0$.

We state that $B = b \cdot G$ for correctness, under the premise of $P_1 = Q'$ in $\mathcal{V}_{N,P}$. We have $B = Q_1 - \mathrm{Hash}(\mathsf{ICert}_1) \cdot P_1 = \left(\mathrm{Hash}(\mathsf{ICert}_1) \cdot d' + b\right) \cdot G - \mathrm{Hash}(\mathsf{ICert}_1) \cdot P_1 = b \cdot G$ If $\mathcal{A}$ successfully constructs the eligible nonroot implicit certificate, $\mathcal{B}$ is also successful in solving the ECDLP problem with the same probability.

References

1. CertVerifyCertificateChainPolicy function (wincrypt.h) (2021). https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-certverifycertificatechainpolicy
2. Certificate key matcher (unknown). https://www.sslshopper.com/certificate-key-matcher.html
3. Administration, C.E.: SM2 elliptic curve public key algorithms (2010)
4. BlackBerry: Certicom device certification authority for zigbee smart energy (nd). https://blackberry.certicom.com/en/products/managed-certificate-service/smart-energy-device-certificate-service
5. Brown, D.R.: SEC 2: Recommended elliptic curve domain parameters. In: Standars for Efficient Cryptography (2010)
6. Brown, D.R., Campagna, M.J., Vanstone, S.A.: Security of ECQV-certified ECDSA against passive adversaries. Cryptology ePrint Archive (2009)
7. Brown, D.R.L., Gallant, R., Vanstone, S.A.: Provably secure implicit certificate schemes. In: Syverson, P. (ed.) FC 2001. LNCS, vol. 2339, pp. 156–165. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46088-8_15
8. Brubaker, C., Jana, S., Ray, B., Khurshid, S., Shmatikov, V.: Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In: 2014 IEEE Symposium on Security and Privacy, pp. 114–129. IEEE (2014)
9. Campagna, M.: SEC4: Elliptic curve Qu-Vanstone implicit certificates, version 1.0. Tech. rep., Standards for Efficient Cryptography (2013)
10. ETSI, T.: ETSI TS 103 097 v1.1.1-intelligent transport systems (ITS); security; security header and certificate formats. Standard, TC ITS (2013)
11. Ford, W., Poeluev, Y.: The machine-to-machine (M2M) public key certificate format. Internet-Draft draft-ford-m2mcertificate-00, IETF Secretariat (2015)
12. Forum, N.: Signature record type definition, technical specification, v2.0 (2014)
13. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: validating SSL certificates in non-browser software. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 38–49 (2012)
14. IEEE 1609 Working Group and others: IEEE standard for wireless access in vehicular environments-security services for applications and management messages. IEEE STD 1609(2) (2016)
15. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (2001)
16. Labs, M.: What CVE-2020-0601 teaches us about Microsoft's TLS certificate verification process (2020). https://www.mcafee.com/blogs/other-blogs/mcafee-labs/what-cve-2020-0601-teaches-us-about-microsofts-tls-certificate-verification-process/
17. Mavrogiannopoulos, N., Vercauteren, F., Velichkov, V., Preneel, B.: A cross-protocol attack on the TLS protocol. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 62–72 (2012)
18. Msahli, Cam-Winget, W.: Internet X.509 public key infrastructure certificate. Tech. rep., RFC 8902 (2020)
19. National Security Agency: Patch critical cryptographic vulnerability in Microsoft windows clients and servers (2020). https://media.defense.gov/2020/Jan/14/2002234275/-1/-1/0/CSA-WINDOWS-10-CRYPT-LIB-20190114.PDF

20. Paganini, P.: Two PoC exploits for CVE-2020-0601 nsacrypto flaw released (2020). https://securityaffairs.co/wordpress/96486/uncategorized/cve-2020-0601-nsacrypto-exploits.html
21. Poeluev, Y., Ford, W.: Transport layer security (TLS) and datagram transport layer security (DTLS) authentication using m2m certificate. IETF Secretariat (2015)
22. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_33
23. Polk, T., Housley, R., Bassham, L.: Algorithms and identifiers for the internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. Tech. rep., RFC 3279 (2002)
24. Pollicino, F., Stabili, D., Ferretti, L., Marchetti, M.: An experimental analysis of ECQV implicit certificates performance in VANETs. In: 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), pp. 1–6. IEEE (2020)
25. Qi'an Xin Codesafe: Detailed analysis of CVE-2020-0601 vulnerability (in Chinese) (2020). https://blog.csdn.net/smellycat000/article/details/104057852
26. Romailer, Y.: CVE-2020-0601: The Chainoffools/Curveball attack explained with POC (2020). https://research.kudelskisecurity.com/2020/01/15/cve-2020-0601-the-chainoffools-attack-explained-with-poc/
27. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
28. Simpson, J.: A technical analysis of Curveball (cve-2020-0601) (2020). https://www.trendmicro.com/en_us/research/20/b/an-in-depth-technical-analysis-of-curveball-cve-2020-0601.html
29. Wagner, D., Schneier, B., et al.: Analysis of the SSL 3.0 protocol. In: The Second USENIX Workshop on Electronic Commerce Proceedings, vol. 1, pp. 29–40 (1996)
30. Whyte, W., Weimerskirch, A., Kumar, V., Hehn, T.: A security credential management system for V2V communications. In: 2013 IEEE Vehicular Networking Conference, pp. 1–8. IEEE (2013)