INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2025, VOL. 71, NO. 2, PP. 453–461
Manuscript received February 21, 2025; revised May, 2025. doi: 10.24425/ijet.2025.153592

On cofactored verification of EdDSA signatures

Adrian Cinal, Oliwer Sobolewski

***Abstract*—EdDSA is a Schnorr signature scheme instantiated on top of Edwards curves, which admit fast, constant-time arithmetic, but suffer from the presence of a non-trivial cofactor, where the order of the group of points is a large prime times a small integer (4 or 8). Current standards permit for points present in the signature (commitment and/or public key) to have a component in the small-order subgroup of the group of points. This is done by sanctioning two variants of the signature verification equation and specifying precedence of one over the other. This last point, however, seems to be widely misunderstood and the two variants are given equal footing, allowing different "compliant" implementations to use different verification algorithms. This in turn lets malicious actors create signatures which are accepted by some parties, but rejected by others, threatening, e.g., consensus in a blockchain network setting. We add to the discussion on practical consequences of such discrepancies by formulating the consensus problem in the context of load-shedding attacks. We argue that the standards *are* in fact very specific about the set of valid signatures, despite lacking in explicitness and emphasis. We further show that two mainstream cryptographic libraries, namely, OpenSSL and CIRCL, accidentally (and in a manner not immediately apparent when inspecting the code) use the correct variant of the verification equation for one parameter set of EdDSA, but incorrect for another. In OpenSSL, this is traced back to careless copying of refcode. We conclude by proposing remedies to the chaotic status quo described.**

***Keywords*—cryptographic standards, cryptographic implementations, consensus, cofactor**

I. Introduction

EdDSA is a modern instance of the Schnorr signature scheme with the underlying group structure afforded by the points on a *twisted Edwards curve*. Whereas Schnorr required the group to have prime order, the number of points on an Edwards curve is always a multiple of four. This gives rise to a small subgroup of order 4 and 8 in the case of, respectively, Ed448 and Ed25519, the standard parameter sets for EdDSA. Signatures produced according to [1] and [2] (which we henceforth refer to collectively as *the standards*[1]) should use only points lying in the large prime-order subgroup. Malicious actors have more freedom, however, in their choice of points, since checking membership of points in the large subgroup is not enforced by the standards and is typically not done by implementations. (Membership is explicitly tested in some delicate contexts where Edwards curves are used such as in Monero, see [3],[2] but not in plain EdDSA. Checking membership naïvely can be costly, and little effort has been put into optimizing it; see, e.g., [4].)

This raises the question on how such maliciously constructed points should be treated. Should the Schnorr verification take place in the group of points on the curve or their quotients modulo the small subgroup, i.e., should two points be considered equivalent if they differ by, say, a point of order 2? The standards are not explicit enough in this regard and seemingly endorse both variants (but, as we argue in Section VI, the intention was different). Importantly, the set of signatures verifiable using the latter group is strictly larger than the set of signatures verifiable using the former (which corresponds to the outputs of the "canonical" signature creation algorithm prescribed in the standards). Sanctioning two variants of the verification equation enables malicious actors to issue signatures that are accepted under one but rejected under the other, thereby leading to disagreements between different "compliant" implementations. This has been alluded to in earlier works [5]–[7], but, to the best of our knowledge, never systematically reviewed.

Contributions

In this work, we give a comprehensive account of the issues related to Schnorr signature verification in a group with a non-trivial cofactor and discuss their practical consequences in the setting of distributed protocols. We point out what we believe to be a misphrasing on the part of the EdDSA standards and argue that most mainstream cryptographic libraries actually fail to comply with these standards. Those that do comply, do so inconsistently across parameter sets and only by accident, due to an undocumented optimization that they employ. We trace back the presence of this optimization in OpenSSL's implementation of Ed448 to the careless copying of Hamburg's refcode [8].

II. Preliminaries

A. Edwards Curves

In [9], Edwards introduced his "normal" form of elliptic curves, later adapted by Bernstein and Lange in [10] to the computational setting by proposing efficient addition formulae. Then, in [11], Bernstein et al. generalized the notion of an Edwards curve to what is known as a *twisted Edwards curve*.

A. Cinal and O. Sobolewski are with the Department of Cryptology, NASK National Research Institute, Warsaw, Poland (e-mail: {adrian.cinal, oliwer.sobolewski}@nask.pl).

[1]This is not technically accurate, since RFC 8032 is an Informational RFC, not a standards-track document. We choose to treat it as such, however, since it is viewed as authoritative by software engineers providing the actual implementations, and it is the implementations, and the errors made therein, that we are most concerned with in this paper.

[2]This was not always the case and lead to a double-spending vulnerability; see https://www.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html.

Going forward, we shall abandon the distinction and only work with the more general form of *twisted* Edwards curves, and instead reserve the terms *twisted* and *untwisted* to mean what they mean in [8], which we recall below. Thus, in general, an Edwards curve E over a field $\mathbb{K}$ of odd characteristic is defined by the equation:

$$ax^2 + y^2 = 1 + dx^2y^2\,, \tag{1}$$

where d is a non-square in the field, and a is a square. We denote the set of ($\mathbb{K}$-rational) *points* $(x, y) \in \mathbb{K} \times \mathbb{K}$ satisfying Equation (1) by $E(\mathbb{K})$. For an appropriately defined map $+ : E(\mathbb{K}) \times E(\mathbb{K}) \to E(\mathbb{K})$, the set $E(\mathbb{K})$ forms an abelian group.

In EdDSA, we shall have $\mathbb{K}$ be a prime field $\mathbb{F}_p$ for a large prime p, and $a \in \{-1, 1\}$. The generalization from the curves as originally introduced by Edwards [9], [10] to the twisted form [11] amounts to the introduction of the a parameter; hence, curves with $a = 1$ are sometimes referred to as *untwisted*, and those with $a = -1$ are explicitly said to be *twisted* (cf. [8]). We follow this naming convention in this paper. Notably, twisted Edwards curves ($a = -1$) admit faster algorithms for point addition, introduced by Hisil et al. in [12]. For $p \equiv 3 \pmod 4$, however, $a = -1$ is not a square, limiting the use of twisted Edwards curves over $\mathbb{F}_p$. This point shall be important and shall be taken up again below.

Edwards curves are attractive for cryptographic use because of their simple and symmetric group law:

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}\right),$$

which, for d and a satisfying the conditions above, is *complete*, i.e., has no exceptions among the rational points on the curve. It is also *unified*, in that it is valid in the case where $(x_1, y_1) = (x_2, y_2)$ (point *doubling*). The neutral element under this addition law is $(0, 1)$, and a negative (additive inverse) of a point (x, y) is the point $(-x, y)$.

Note, however, that, for Edwards curves over $\mathbb{F}_p$, the number of rational points is divisible by four, whereas for cryptographic purposes we typically want groups to have prime order (cf. Schnorr groups). For curves used in practical applications, we have $\#E(\mathbb{F}_p) = cq$, where $\#E(\mathbb{F}_p)$ denotes the number of $\mathbb{F}_p$-rational points on a curve E, q is a large prime, and c is a small *cofactor*, either 4 or 8, depending on the curve. We shall normally restrict all computations to be done in the *q-torsion* of $E(\mathbb{F}_p)$, i.e., the cyclic subgroup of order q, denoted $E(\mathbb{F}_p)[q]$. From a security standpoint, this restriction is immediate since the discrete logarithm problem in a composite-order group is only as hard as in its largest cyclic subgroup. Coincidentally, it also facilitates using twisted curves over fields where $a = -1$ is not a square (i.e., $p \equiv 3 \pmod 4$), namely, even in that case, the addition formulae remain complete for points in the q-torsion of $E(\mathbb{F}_p)$, so one way to ensure completeness of the group law is to restrict the computations to $E(\mathbb{F}_p)[q]$ (cf. [8]). Importantly, however, malicious actors need not follow this restriction and may use points outside this group; specifically, they may add *small-order* components from $E(\mathbb{F}_p)[c]$ (the c-torsion of $E(\mathbb{F}_p)$) to the points they publish. We follow [6] and refer to points P that have order $\mathrm{ord}(P) > q$, i.e., have a non-trivial component in the c-torsion, as being *mixed-order*.

B. EdDSA

In [13], Bernstein et al. introduced EdDSA, a modern, Edwards curves-based instance of the Schnorr signature scheme. EdDSA, while attractive for a number of reasons, suffers from the issue of the cofactor. Specifically, there exists a non-trivial subgroup $E(\mathbb{F}_p)[c]$ of points outside the normal domain of computation which is the prime-order subgroup (the q-torsion). The original designers recognized it and, in [13], proposed two variants of the verification equation, showing that neither gives an adversary any non-negligible advantage in forging signatures. Their treatment of the two variants can be viewed as due scientific diligence, exploring the potential pitfalls that implementers may find themselves stepping into.

The standards mention both variants, but do not give them equal footing and instead favour, as we argue in Section VI, one above the other. The way the standards phrase this, however, has apparently confused most implementers, opening a pathway to different implementations disagreeing about the validity of signatures. We shall argue below that the standards are written in a way that is actually not ambivalent but nevertheless difficult to parse and comprehend.

Let B be the distinguished *basepoint* that generates $E(\mathbb{F}_p)[q]$. A signature $\sigma = (R, s) \in E(\mathbb{F}_p) \times \mathbb{Z}_q^*$ on a message M and under a public key $A \in E(\mathbb{F}_p)$ is accepted (verifiable) if (but not "only if," as we shall argue)

$$[s]B = R + [h]A\,, \tag{2}$$

where $h = \mathsf{Hash}(R, A, M)$. Equation (2) corresponds to *cofactorless verification*. Let T be a point in $E(\mathbb{F}_p)[c]$ distinct from the identity.[3] Note that if $R = [r]B + T$, but $A \in E(\mathbb{F}_p)[q]$, Equation (2) can never hold. The standards, however, "permit" implementations to instead check that

$$[c][s]B = [c]R + [c][h]A\,, \tag{3}$$

which we refer to as *cofactored verification*. If $R \in E(\mathbb{F}_p) \setminus E(\mathbb{F}_p)[q]$ as above, Equation (3) still holds (assuming s is computed correctly), whereas Equation (2) does not. This is because cofactored verification tests not for equality between Edwards curve points, but between equivalence classes of points modulo the c-torsion $E(\mathbb{F}_p)[c]$. Legitimate signatures, produced according to the specification [1], [2], pass both cofactorless and cofactored verification, and we shall refer to those that pass only cofactored verification as *contentious signatures*. For typical applications, the more restrictive (and slightly faster) cofactorless verification is usually preferred (see [6]). We shall argue in Section VI that this is actually not compliant with the standards.

We remark here on the similarity of this problem and *algorithm substitution attacks*. For legitimate signatures, two implementations using, respectively, Equation (2) and Equation (3) are indistinguishable, but for adversarial inputs, namely, contentious signatures, their behaviours diverge. We

[3] In the case of Ed25519, a standard instance of EdDSA, $E(\mathbb{F}_p)[c]$ is cyclic, so a T exists that generates the whole subgroup.

elaborate in Section IV on the practical consequences of this behaviour.

C. Blockchain Networks

Since contentious signatures threaten consensus, we now turn to a principal application of consensus protocols, namely, blockchain networks. A *blockchain network* is a peer-to-peer network of nodes working together to create a chain of *transactions*, grouped in *blocks* tied together with cryptographic hashes — a *blockchain*. The transactions express changes to a global state of the network, e.g., the distribution of money in the case of cryptocurrencies. In most settings, a transaction must be authenticated by means of a digital signature by its creator. Validity of this signature, together with the consistency of other data in a transaction (e.g., spent amount resulting in a non-negative balance of a user's cryptocurrency account), forms the basis of consensus. Indeed, a fundamental assumption in the analysis of blockchain networks is that participants in the network agree on what constitutes a valid transaction and a valid block. They may, however, disagree about what the most recent blocks of the chain are. Due to the distributed nature of the system, different nodes see different blocks first and append them to the tip of the chain. When two conflicting chains arise, nodes are required by the protocol to reject the shorter chain. It can then be argued that, under normal operating conditions, the network should eventually reach consensus, and all such *forks* are transient.

A number of consensus mechanisms exist with the most common being *Proof of Work*. In a Proof-of-Work blockchain, blocks get *mined* by a portion of the network called *miners*, who (for a reward) solve cryptographic puzzles associated with the blocks. The difficulty of the puzzles is set by the protocol in proportion to the *hashrate* of the network, which is the capability of the network for solving puzzles, expressed in solution attempts (hashes computed) per unit of time, but the cost of verifying a puzzle solution is constant and small. The use of cryptographic puzzles is necessary in cryptocurrency blockchains to prevent *double-spending*, i.e., creation of an alternate chain in which a given transaction was not included and convincing the network to adopt this chain, thereby making the funds previously spent in a transaction eligible for spending again. Specifically, the requirement to solve a puzzle before publishing a transaction protects against Sybil attacks [14], where an adversary creates multiple accounts to overwhelm the network, since, now, significant computational resources must be invested to participate in the protocol.

An adversary that has enough computing power to outpace the honest nodes, i.e., controls more than 50% of the miners, can mutate the blockchain at will, since they can always build the longest chain, which the network must accept as per the protocol. This attack, termed a 51% *attack*, in practice, compromises the blockchain.[4]

Importantly, *all* consensus mechanisms, Proof of Work included, assume that invalid blocks will be rejected by honest nodes, and thus, if the majority of the nodes are honest, only valid blocks will be added to the chain. If there is a disagreement between honest nodes about block validity, the security assumptions for the blockchain protocol are broken, possibly invalidating further security claims.

[4]For examples of successful attacks, see, e.g., the Bitcoin Gold or Ethereum Classic cases. Importantly, the 51% attack is also applicable to the Proof of Stake consensus model. There, the adversary needs to control more than 50% of the staked currency.

III. Hamburg's 4-Isogeny

To address the slowdown incurred by using untwisted ($a = 1$) curves on top of fields with characteristic $p \equiv 3 \pmod 4$ (where $a = -1$ is not a square), Hamburg proposed in [8] an efficient homomorphism or *isogeny* from such curves (in the case when $\#E(\mathbb{F}_p) = 4q$, i.e., $c = 4$) to *isogenous* twisted ($a = -1$) curves which admit faster arithmetic. Under this isogeny (ϕ_a in [8]), points in $E(\mathbb{F}_p)[c]$ vanish, thus ensuring that all computations on the isogenous curve (twisted) are well-defined and the group law is complete. While for signing purposes this optimization is safe, for verification, it amounts to (apart from performance increase) switching from cofactorless to cofactored verification. Indeed, Hamburg proposes that the verification equation take the form:

$$[s]\phi_a(B) = \phi_a(R) + [h]\phi_a(A)\,. \tag{4}$$

Since ϕ_a is injective on $E(\mathbb{F}_p)[q]$, Equation (4) is equivalent to Equation (3). If we let $R = [r]B + [k]T$ as before, we observe that

$$\phi_a(R) = [r]\phi_a(B) + [k]\phi_a(T) = [r]\phi_a(B)\,,$$

since $T \in E(\mathbb{F}_p)[c] = \ker \phi_a$ vanishes under ϕ_a.

One curve to which Hamburg's 4-isogeny is applicable is Ed448, introduced by him a year later [15] and standardized in [1], [2]. Whereas [15] proposes an implementation explicitly using the Decaf construction [7] to work not with points on Ed448, but equivalence classes modulo $E(\mathbb{F}_p)[c]$, OpenSSL adopts the isogeny from [8] on its own, without mentioning Decaf, while also not documenting it explicitly. So does Cloudflare's CIRCL. Both libraries use cofactorless verification for Ed25519 (another standard instance of EdDSA). Viewing Ed25519 and Ed448 as parameter sets for the abstract scheme that is EdDSA (see [16]), we believe this to be an implementation error that, for different scheme parameters, different (and incompatible) verification equations are used. Since the standards [1], [2] are vague, however, both variants are believed to be compliant. We argue otherwise. Importantly, engineers and practitioners without background in elliptic curve theory will not identify the switch to cofactored verification in OpenSSL's and CIRCL's implementations of Ed448 by simply studying the code. The "clearing" of $E(\mathbb{F}_p)[c]$ is not explicit as in Equation (3), but instead obfuscated in the computation of the 4-isogeny.

The likely reason for OpenSSL and CIRCL's mistake (as we perceive it) is that, to accompany [15], Hamburg only provided reference code for Ed448 which already used Decaf. It can be seen in commit history that OpenSSL directly imported Hamburg's refcode,[5] only to drop references to Decaf during

[5]See commit `7324473f893ef135693cf983b415f19a0366d539` (`Import Curve 448 support`).

integration and refactoring.[6] Other libraries (e.g., WolfSSL) built their Ed448 implementations from the ground up by, e.g., adapting Bernstein's refcode for Ed25519.

IV. Practical Consequences

In this section, we show possible issues arising from the ambiguity of signature verification algorithms as implemented in different cryptographic libraries. The problem can have serious consequences when unaccounted for by system developers in practical applications that heavily rely on signatures to, e.g., achieve consensus.

A. Implementation Fingerprinting

One immediate consequence of applying the isogeny optimization in OpenSSL and CIRCL is the existence of an efficient implementation fingerprinting oracle. Specifically, given an API that accepts Ed448 signatures, one can submit signatures (R, s) with $R \notin E(\mathbb{F}_p)[q]$ and observe if they are accepted.[7] If a bug is found in OpenSSL or CIRCL, such an API would allow hackers to quickly identify vulnerable servers. For example, the tuple (M, A, σ) in Figure 1 is verifiable by OpenSSL/CIRCL, but would be rejected by, e.g., WolfSSL. The tuple was obtained from the first Ed448 test vector in [2] by adding the point $(0, -1) \in E(\mathbb{F}_p)[2]$ to the commitment $[r]B$ (and recomputing the response s).

```
message = ""
public_key = \
    "5fd7449b59b461fd2ce787ec616a" + \
    "d46a1da1342485a70e1f8a0ea75d" + \
    "80e96778edf124769b46c7061bd6" + \
    "783df1e50f6cd1fa1abeafe82561" + \
    "80"
signature = \
    "acc5c809441ba8dae0fdc3f27706" + \
    "8951d204afb57bc1cb2df8b027dc" + \
    "2ae5a6e0d4dcc0fcb09d7d7e0d02" + \
    "85dd222b8287d73a642f5de402c6" + \
    "00ddc908c93ed05d2f6048e19804" + \
    "a4d5f9187f23d53298a2527daf61" + \
    "f0d95565eda3cd71cffdc24c31e9" + \
    "74d3ceef1042966ea9cdeb1489e4" + \
    "0c00"
```

Fig. 1. A contentious signature that can be used to distinguish between a verifier running OpenSSL/CIRCL and, say, WolfSSL.

B. Load-Shedding Attack

A fundamental assumption of blockchain networks is that each block published is either perceived as valid by *all* honest nodes or invalid by *all* honest nodes. Lack of explicit standardization of EdDSA verification algorithm undermines this assumption if transactions are signed with EdDSA.

If different miners use different verification variants, transactions with contentious signatures would only be included in blocks of some of the miners. If the majority of the network then used cofactorless verification (Equation (2)), this would mean that some of the miners' efforts were wasted. Similarly, if some miners used cofactored verification (Equation (3)), then malicious actors could publish transactions with contentious signatures, so that the blocks mined by miners running cofactored verification would be rejected by (some fraction of) the network.

We observe that this corresponds to a *load-shedding attack* introduced by Tiwari and Green in [17]. Emitting *contentious transactions* reduces the overall hashrate of the network, so that the attacker now comprises a higher percentage thereof. This facilitates the *selfish mining* strategy [18] and makes forking the blockchain easier. Specifically, for the 51% attack introduced in Section II-C, the adversary now only needs to control more than 50% of one of the subnetworks created by the fork.

For example, if 40% (in terms of hashrate) of the network used WolfSSL, where Equation (2) is used, and 60% used OpenSSL, where Equation (3) is used instead, then an adversary would only need around 30% of the initial network's hashrate to control the OpenSSL-based subnetwork. If only contentious signatures were published, then the WolfSSL-based 40% of the network would effectively not be participating in the protocol, and even just one contentious block accepted and added to the blockchain would make the WolfSSL-based nodes play a losing game, where they are trying to build a longer chain with less than half of the network's computing resources. This idea is illustrated in Figures 2 and 3. Importantly, having 30% of the network's hashrate is not without precedent; in September 2024, the Foundry USA mining pool had 30.7% of the overall network's hashrate.[8]

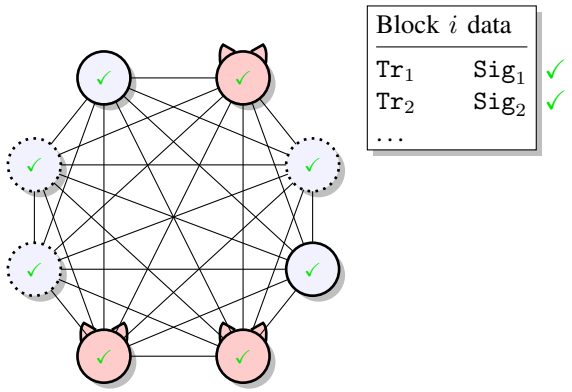

Fig. 2. The original blockchain network, where the adversary controls 3 out of 8 equally powerful nodes. Some nodes (dotted circles) use WolfSSL, and others (solid circles) use OpenSSL. The adversary controls less than 50% of the network's computational power.

Note that most blockchains do in fact support multiple client applications, and, due to distributed trust, it is actually preferred that clients diversify the cryptographic libraries they use (to minimize the chance that a vulnerability in one library

[6] See commit `aeeef83cb536216a414287dee1f424265283da88 (Remove references to libdecaf)`.

[7] Note that R cannot belong entirely to $E(\mathbb{F}_p)[c]$ as per the standards, but can have components in both the q-torsion and the c-torsion.

[8] https://btc.com/stats/pool.

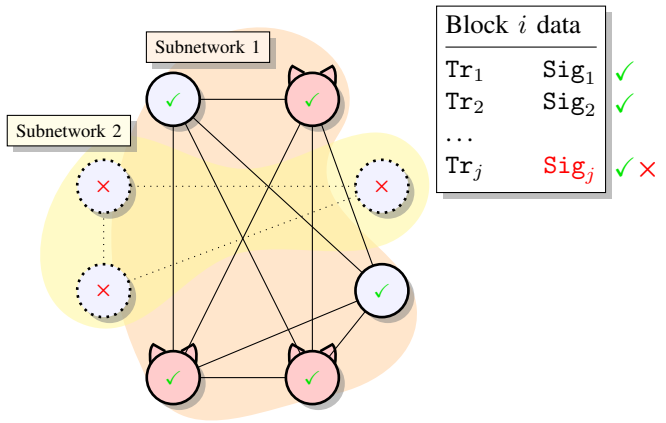

Fig. 3. The network from Figure 2 after introducing a block with a contentious signature $\mathtt{Sig}_j$ with components in $E(\mathbb{F}_p)[c]$. The network forks into two subnetworks: subnetwork 1, where the block is accepted and published, and subnetwork 2, where the block is rejected. The adversary controls 3 out of 5 nodes in subnetwork 1, enabling a 51% attack.

affects the entire network, i.e., becomes a single point of failure).

The problem of verification requirements standards for EdDSA signatures in the cryptocurrency space has been identified before. In particular, the Zcash cryptocurrency addressed the problem in one of the Zcash Improvement Proposals [5] by explicitly requiring that cofactored verification be used. The reasoning behind this, given in [5], was the following:

> *... implementations conformant to RFC 8032 need not agree on whether signatures are valid. This is unacceptable for a consensus-critical application like Zcash.*

While we argue in Section VI that the claim is in fact not entirely correct and being truly conformant to RFC 8032 *does* guarantee consensus, existing implementations do in fact disagree about validity of signatures (R, s) with mixed-order commitments R.

C. Byzantine (Dis)agreements

Consensus protocols in the presence of malicious or faulty nodes, so-called *Byzantine agreement protocols*, are a more general problem, with blockchains being only an example. A protocol that works correctly (i.e., all honest nodes come to a consensus) even when $t > 0$ out of n nodes are malicious is said to be *Byzantine fault tolerant* (BFT) [19], [20].

An upper bound on t for *unauthenticated* schemes is $t \leq \frac{n-1}{3}$ [21], but use of digital signatures and a public key infrastructure enables secure Byzantine agreement protocols for $t \leq \frac{n-1}{2}$ [22]. For this reason, signature schemes are often used in Byzantine consensus protocols, and their main role is to ensure that malicious nodes cannot change the contents of messages sent by honest nodes (without being detected). Observe that, if a Byzantine agreement protocol relies on EdDSA and different nodes use different cryptographic libraries, some of which implement Equation (2), while others implement Equation (3), then an attack similar to the above is possible. Specifically, malicious nodes can introduce contentious signatures into the protocol, which will be successfully verified by some nodes, call them "cofactored nodes," as they use Equation (3), and rejected by others, call them "cofactorless nodes," as they use Equation (2).

Consider a Byzantine agreement protocol and assume that a malicious node only emits contentious signatures (i.e., signs their messages in the protocol with small-order components added to the Schnorr commitments R) and otherwise follows the protocol. At some point in time, the protocol terminates for "cofactored nodes," and all of them agree on the same value, but, for "cofactorless nodes," the protocol has either not terminated yet, or its output is different (most likely) than for the "cofactored nodes." This naturally breaks the functionality of Byzantine agreement. The effectiveness of the attack depends on the ratio of "cofactored" to "cofactorless nodes" and is highest when the split is even, but, formally, the scheme is broken when even one honest node rejects a contentious signature, since a fundamental assumption about consensus protocols is violated.

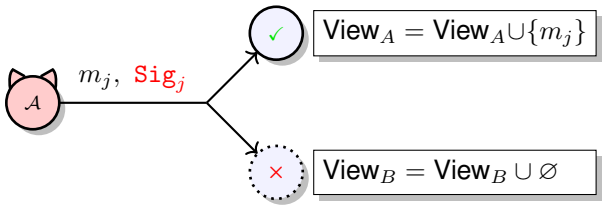

Fig. 4. Let $\mathtt{View}_i = \{m_0, \ldots, m_j\}$ be the view of node i, i.e., the set of all messages it has received during all phases of a Byzantine agreement protocol. Then, a malicious node can produce a contentious signature under an honest message such that it will be verified and added to their view by "cofactored nodes" (solid circle) and rejected by "cofactorless nodes" (dotted circle). The difference in views may lead to different outcomes of the agreement protocol for the two nodes.

D. Breaking Decentralized Electronic Voting

Another set of consensus-critical protocols can be found in electronic voting (*e-voting*). E-voting is a system that allows a set of eligible users to make a decision based on individual votes and must meet following security requirements [23]:

1) **(Privacy)** votes are anonymous,
2) **(Fairness)** no result is obtainable before the end of the voting process,
3) **(Eligibility)** only eligible users can vote at all, and they can vote exactly once,
4) **(Coercion-resistance)** an eligible user can cast their vote despite outside influence,
5) **(Individual verifiability)** every user has the ability to check whether their votes were counted,
6) **(Universal verifiability**[9]**)** everyone has the ability to validate the voting outcome,

There are many cryptographic methods of constructing e-voting systems, e.g., based on mix-nets, homomorphic encryption, blind signatures, or blockchains (for a systematic

[9]Required in decentralized e-voting systems.

review, see [24]). In our case, the most interesting ones are the blockchain-based schemes (e.g., [23]) due to their distributed nature. These systems use a blockchain to store ballots so that votes, once cast, cannot be deleted or altered. Every node (both user and voter) is responsible for rejecting fraudulent votes (analogously to how invalid transactions are rejected in cryptocurrencies), so that consensus can be maintained in accordance with the voting rules.

If eligible but malicious nodes inject votes with contentious signatures on them (in the same manner as in Section IV-B), the result of the vote (in particular, the vote count) will be different depending on whether a particular node runs a cryptographic library that implements Equation (2) or Equation (3). The honest nodes' decision is then not unanimous, breaking the requirement of universal verifiability.

V. Advantages of Cofactored Verification

All signatures *produced* as prescribed in the standards [1], [2] are verifiable using Equation (2). This would suggest that *cofactorless* verification, which is also faster, albeit marginally, should be the norm. This is in line with a conservative approach that is a hallmark of computer security. Notwithstanding, cofactored verification does come with a number of advantages which we summarize in this section.

A. Batch Verification

The cost of single-signature verification may be amortized by using *batch verification*. Batch verification, already proposed in [13], exploits the linearity of the Schnorr verification equation by checking it not with a single tuple (R, h, s), but instead with a random integer linear combination of many such tuples corresponding to a *batch* of signatures $\{(R_i, s_i)\}_i$ on messages $\{M_i\}_i$ under public keys $\{A_i\}_i$:

$$\left[\sum_i z_i s_i\right] B = \sum_i [z_i] R_i + \sum_i [z_i h_i] A_i \,. \tag{5}$$

The z_i's in Equation (5) are drawn uniformly at random from $\mathbb{Z}_q^*$, and $h_i = \mathsf{Hash}(R_i, A_i, M_i)$ for each i. Batch verification amortizes the amount of computation done in the group $E(\mathbb{F}_p)$ by trading it for some extra work in the scalar field $\mathbb{Z}_q$, where the operations are much cheaper. Specifically, once the scalars z_i, $\sum_i z_i s_i \pmod q$, and $\sum_i z_i h_i \pmod q$ have been computed, verifying Equation (5) can be done using a multi-scalar multiplication technique that uses a constant number of point doublings regardless of the size of the batch.

Notably, however, performing the scalar computations in $\mathbb{Z}_q$ implicitly assumes that the points R_i and A_i have order q, i.e., lie in $E(\mathbb{F}_p)[q]$. As observed by Chalkias et al. in [6] and (apparently independently) in [5], batch verification is not *compatible* with single-signature verification when the commitments R_i and/or public keys A_i have small-order components. This is because, with non-negligible probability, a signature that would be rejected if verified individually using cofactorless verification (Equation (2)) gets accepted when verified as part of a batch using Equation (5). Indeed, the small-order components of the commitments R_i or public keys A_i may vanish under the multiplication by z_i or cancel with small-order components (scaled by the random z_i's) of other points. Since the order of any small-order point $T \in E(\mathbb{F}_p)[c]$ divides the cofactor c by Lagrange's theorem, the probability of such cancellations is non-negligible. Notably, such discrepancies can occur *within* a single software library, not necessarily between distinct implementations, i.e., the same signature could be accepted or rejected by the same library depending on whether it was verified individually or in a batch (and depending on the choice of the random scalars z_i in batch verification, making batch verification non-deterministic).

If a "cofactored variant" of Equation (5) is used, i.e., both sides are multiplied by c, then batch verification becomes deterministic and compatible with single-signature verification (using Equation (3)) in that any signature accepted as part of a batch is accepted (except with negligible probability) when treated individually.

B. Reducing the Number of Point Doublings

Antipa et al. in [25] observed that ElGamal-like signatures can be verified efficiently by considering, instead of the original verification equation (in the relevant group), a non-zero scalar multiple thereof. For an appropriate choice of this scalar, the number of point doublings (squarings) in the double-and-add (square-and-multiply) algorithm or one of its many variations. Their result naturally applies to Schnorr signatures as well, so to EdDSA in particular. Consider again the Schnorr verification equation. If it holds, then so does any non-zero scalar multiple of it (indeed, this implication goes both ways if the underlying group $E(\mathbb{F}_p)$ is of prime order). Let $z \in \mathbb{Z}_q^*$ and consider the verification equation

$$[zs]B = [z]R + [zh]A \,, \tag{6}$$

where the scalars are implicitly reduced modulo q before multiplying the points. Antipa et al. noted that, if we can choose z so that both z and $zh \pmod q$ are "small" as integers, one can greatly optimize signature verification. Specifically, if we can find z such that both z and $zh \pmod q$ have bit lengths approximately half that of q, then we can cut the number of point doublings required to check Equation (6) in half. The length of $zs \pmod q$ is irrelevant. Indeed, since B is a fixed parameter of the scheme, implementations may precompute $\tilde{B} = [2^{\lambda/2}]B$, where $\lambda = |q|$, and split $e = zs \pmod q$ as an integer into two parts e_0 and e_1 such that $e = e_0 + 2^{\lambda/2} e_1$. Signature verification then proceeds by checking that

$$[e_0]B + [e_1]\tilde{B} - [z]R - [zh]A = (0, 1) \,, \tag{7}$$

where all the scalars have bit lengths approximately $\lambda/2$ and the cost of point doublings (the number of which has already been cut in half) can be amortized by using multi-scalar multiplication.

The construction of Antipa et al. has already been considered in [13] and rejected as the overhead was deemed too large. Pornin in [26], however, proposed an efficient, lattice-based algorithm for finding the scaling factor z, thus making the optimization worthwhile. Pornin also discusses in his work the issue of the cofactor and rightly remarks that rescaling the verification equation by z may lead to small-order components

vanishing by virtue of z or $zh \pmod q$ being a multiple of the cofactor (or just the order of the small-order component). Thus, verifiers using the optimization of Antipa et al. may accept signatures that would otherwise be rejected, even without explicitly clearing the small-order components via multiplication by c. Settling on using cofactored verification (Equation (3)) again avoids this problem.

VI. Standardization of EdDSA

As pointed out already, introduction of two verification equations by the creators of EdDSA should be viewed as scientific diligence. [13] reads:[10]

> *The verifier is* permitted *to check this stronger equation and to reject alleged signatures where the stronger equation does not hold. However, this is not* required*; checking that* $[8][s]B = [8]R + [8][h]A$ *is enough for security.*

Specifically, it says that implementations are permitted to reject signatures failing to satisfy Equation (2).

Bernstein et al. in [13] are indeed ambiguous about which verification equation is to be preferred and do not make an attempt at normalization. Standards' writers, however, must have recognized the dangers of consensus underspecification and write (in both [2] and later [1]) that cofactored verification equation (Equation (3)) should be checked and that it is *sufficient, but not required* to instead check Equation (2). Specifically, FIPS Digital Signature Standard [1] reads:

> *Check that the verification equation* $[cs]B = [c]R + [ch]A$ *holds. It's sufficient, but not required, to instead check* $[s]B = R + [h]A$*. Output "reject" if verification fails; output "accept" otherwise.*

Similarly, RFC 8032 [2] reads for Ed25519:

> *Check the group equation* $[8][s]B = [8]R + [8][h]A$*. It's sufficient, but not required, to instead check* $[s]B = R + [h]A$*.*

For Ed448, [2] says the same thing except 8 is replaced with 4.

We note that the Initial Public Draft (IPD) [27] of [1] originally read:

> *Check that the verification equation* $[cs]B = [c]R + [ch]A$ *holds. Output "reject" if verification fails; output "accept" otherwise.*

No mention was made of cofactorless verification whatsoever. It was only changed after a comment from Ruggero Susella [28]:

> *. . . this definition seems to disallow the use of the more commonly used cofactorless verification* $[s]B = R + [h]A$*. Cofactorless verification is significantly simpler than cofactor verification, avoiding the need of decompressing* R*, to multiply by the cofactor and to convert* $[c]R$ *back to affine coordinates. Therefore, we would prefer that FIPS 186-5 will more clearly allow, as RFC 8032 does, the usage of cofactorless verification. In case the intention was to remove ambiguity between the two verifications by defining only one, we would favor cofactorless verification over cofactor verification, as XEdDSA did, because we do not see meaningful advantages to justify the additional cost.*

[10] Notation adapted to ours both here and later in the text. Emphasis as in [13].

We argue against Susella's argument. It is true that R must be decompressed, but otherwise $[s]B - [h]A$ needs to be *compressed* (or at least converted to affine coordinates) to check Equation (2). Compression has cost largely comparable with decompression (one requires an inversion in the field, the other an exponentiation — both approximately as expensive). Furthermore, if any of the optimizations of Section V are to be employed, R has to be decompressed anyway. We argue that the optimizations known in the literature more than make up for the cost of decompression. The extra multiplication by the cofactor is cheap, as the cofactor is always a small power of two in the standard instances of EdDSA: in the case of Ed448, the multiplication by c amounts to two point doublings and, in the case of Ed25519, to three. Finally, the claim that $[c]R$ must be converted back to affine coordinates for verification purposes is wrong. Equality between points can be checked directly in projective representation.

We also do not believe that RFC 8032 [2] endorses the use of cofactorless verification as understood by Susella. Indeed, it says (as does the final version of FIPS DSS [1]) that "it's sufficient, but not required, *to check*" the stronger equation. We argue this is a misphrasing on the part of the document's authors, since the phrases "sufficient to check" or "required to check" are not meaningful in this context. Indeed, the "check" in question may have different outcomes: the equation may hold or not, and, in each case, the meaning of "sufficient" and "required" is different. We believe the authors instead meant to write: "it's sufficient, but not required, *for the stronger equation to hold*." This is well-defined and in concert with the original intention of at least the FIPS DSS authors (see [27]). If this interpretation is correct, the standards actually give clear guidelines (as standards should) on how EdDSA verification should proceed. Indeed, they spell out an implication (but not an equivalence):

$$\text{Equation (2) holds} \implies \sigma \text{ is correct}.$$

Importantly, however, Equation (2) not holding does not an invalid signature make. Equation (2) is a *sufficient* condition for the validity of σ, but not a *required* one. Checking Equation (2) can only be used in a "happy path" of an EdDSA implementation. Should it fail to hold, implementations should be obligated to fall back to checking Equation (3), which is the "source of truth" about signature validity.

VII. Resolution

We recommend that the standards [1], [2] be revised to more explicitly define EdDSA signature verification rules, similarly to how ZIP 215 [5] does it, to make it clear that *only* cofactored verification is sanctioned, but implementations may *accept* signatures based on Equation (2) holding. It is incorrect to *reject* a signature if Equation (2) does not hold, and Equation (3) must be consulted in that case.

However, since cofactorless verification has its advantages too, being faster and more conservative, we propose standardizing it alongside EdDSA as a separate algorithm and formally distinguishing between the two variants (similarly to how the *prehashed* variant of EdDSA is differentiated from regular

EdDSA [1], [2]). We believe this can be done within the ISO/IEC OID framework by assigning to them distinct *object identifiers* [29].

We introduce *StrictEdDSA* which differs from EdDSA defined in [1], [2] in that cofactorless verification *must* be used. We proceed to give further restrictions on StrictEdDSA. First, no optimization techniques that change the order of points on a non-prime-order curve E are allowed. This includes the optimization by Antipa et al. [25] refined by Pornin [26], batch verification, and using Hamburg's 4-isogeny [8]. Second, note that with each EdDSA signature $\sigma = (R, s)$, associated are *two* "adversarial" points, i.e., points under the control of an adversary, namely, the commitment R and the public key A. It is possible that neither point lies in the q-torsion $E(\mathbb{F}_p)[q]$ and yet Equation (2) holds. This can happen if the small-order components of R and $[h]A$ cancel. So as to effectively enforce that *all* points lie in the q-torsion, implementations *should* check (in addition to checks defined in [2]) that $[q]A = (0, 1)$, the identity in $E(\mathbb{F}_p)$. If the public key is cached and reused across multiple signature verifications, this extra cost gets amortized. We remark, however, that the consensus assumption, that any signature is either valid as per the standard (here, the StrictEdDSA standard) for all nodes or invalid for all nodes, will never be violated even if this check is omitted.

We propose assigning the following *object identifiers* [29] to StrictEdDSA:[11]

```
id-StrictEdDSA25519    OBJECT IDENTIFIER ::= { 1 3 101 116 }
id-StrictEdDSA448      OBJECT IDENTIFIER ::= { 1 3 101 117 }
id-StrictEdDSA25519-ph OBJECT IDENTIFIER ::= { 1 3 101 118 }
id-StrictEdDSA448-ph   OBJECT IDENTIFIER ::= { 1 3 101 119 }
```

where "-ph" denotes the prehashed variants of StrictEdDSA, derived naturally from the corresponding variants of regular EdDSA.

VIII. Related Work

Below, we summarize related work on the subject of EdDSA standardization and deployment, as well as consensus.

Chalkias et al. in [6] point to the discrepancy between software libraries and the FIPS DSS draft [27] and recognize other shortcoming of both the standards and the implementations, giving a comprehensive overview of Ed25519 in the wild. Notably, the authors interpret RFC 8032 [2] as "allowing an optionality" in the choice of verification equation.

Brendel et al. in [30] study different variants of EdDSA and checks performed on the signature values therein. Little emphasis is placed on the issue of cofactored versus cofactorless verification.

Hamburg in [7] proposes Decaf, a point compression format that naturally (and efficiently) does away with small-order components.

Tiwari and Green in [17] study how hardware Trojans in mining rig can be used to launch, among others, a load-shedding attack.

[11] See http://www.oid-info.com/get/1.3.101.

IX. Conclusions

Standardization is a cornerstone of consensus in distributed systems, and disagreements about what constitutes a valid digital signature threaten this consensus. If we, as a community, are not able to decide on how EdDSA signatures are to be verified, there can be no sound reasoning about the security or correctness of distributed protocols relying on EdDSA.

Indeed, the (perceived) ambiguity in the standards leads to new vectors of attack. We have shown, in Section IV-B, how a load-shedding attack, similar to the one from [17], can be launched using contentious EdDSA signatures. Our attack is weaker than the one in [17], but has the advantage of being practically realizable given the current state of both the standardization of EdDSA and the implementations available.

The cofactored versus cofactorless debate must be put to rest, and we must either adopt solutions like Decaf [7] in place of plain EdDSA or settle on one of the verification variants. We argued in this work that *cofactored* verification should be recognized as standard due to its numerous benefits, the most significant of which is compatibility with batch verification (see Section V-A), which already sees use in numerous applications (and was, in fact, already proposed in the original Ed25519 paper [13]). We also argued that the standards [1], [2] *do* actually enforce the use of cofactored verification, but the requirement is poorly phrased and thus widely misunderstood; see Section VI. Finally, we offered a practical solution to the problem; see Section VII.

We hope for our work to engage the community in a discussion on EdDSA standardization and lead to changes in how the specification is formulated.

References

[1] "Digital signature standard," National Institute of Standards and Technology, Federal Information Processing Standards Publication 186-5, 2023. [Online]. Available: {https://csrc.nist.gov/pubs/fips/186-5/final}

[2] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (EdDSA)," Internet Research Task Force, Informational 8032, 2017. [Online]. Available: {https://www.rfc-editor.org/rfc/rfc8032.txt}

[3] Koe, K. M. Alonso, and S. Noether, "Zero to Monero: Second edition," 2020. [Online]. Available: https://www.getmonero.org/library/Zero-to-Monero-2-0-0.pdf

[4] T. Pornin, "Point-halving and subgroup membership in twisted Edwards curves," Cryptology ePrint Archive, Paper 2022/1164, 2022, https://eprint.iacr.org/2022/1164. [Online]. Available: https://eprint.iacr.org/2022/1164

[5] "ZIP 215: Explicitly defining and modifying Ed25519 validation rules," https://zips.z.cash/zip-0215.

[6] K. Chalkias, F. Garillot, and V. Nikolaenko, "Taming the many EdDSAs," Cryptology ePrint Archive, Paper 2020/1244, 2020.

[7] M. Hamburg, "Decaf: Eliminating cofactors through point compression," Cryptology ePrint Archive, Paper 2015/673, 2015. [Online]. Available: https://eprint.iacr.org/2015/673

[8] ——, "Twisting Edwards curves with isogenies," Cryptology ePrint Archive, Paper 2014/027, 2014. [Online]. Available: https://eprint.iacr.org/2014/027

[9] H. M. Edwards, "A normal form for elliptic curves," *Bulletin of the American Mathematical Society*, vol. 44, no. 3, pp. 393–422, 2007. [Online]. Available: https://www.ams.org/journals/bull/2007-44-03/S0273-0979-07-01153-6/S0273-0979-07-01153-6.pdf

[10] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Advances in Cryptology – ASIACRYPT 2007*, ser. Lecture Notes in Computer Science, vol. 4833. Springer, 2007, pp. 29–50.

[11] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards curves," Cryptology ePrint Archive, Paper 2008/013, 2008.

[12] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson, "Twisted Edwards curves revisited," in *Advances in Cryptology - ASIACRYPT 2008*, ser. Lecture Notes in Computer Science, vol. 5350. Springer, 2008, pp. 326–343.
[13] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.
[14] J. R. Douceur, "The Sybil attack," in *International Workshop on Peer-to-Peer Systems*, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:5310675
[15] M. Hamburg, "Ed448-Goldilocks, a new elliptic curve," Cryptology ePrint Archive, Paper 2015/625, 2015. [Online]. Available: https://eprint.iacr.org/2015/625
[16] D. J. Bernstein, S. Josefsson, T. Lange, P. Schwabe, and B.-Y. Yang, "EdDSA for more curves," Cryptology ePrint Archive, Paper 2015/677, 2015. [Online]. Available: https://eprint.iacr.org/2015/677
[17] P. R. Tiwari and M. Green, "Subverting cryptographic hardware used in blockchain consensus," in *Financial Crypto 2024*. International Financial Cryptography Association, 2024.
[18] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Crypto 2014*. International Financial Cryptography Association, 2014.
[19] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, Jul. 1982. [Online]. Available: https://doi.org/10.1145/357172.357176
[20] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, no. 2, p. 228–234, Apr. 1980. [Online]. Available: https://doi.org/10.1145/322186.322188
[21] M. J. Fischer, N. A. Lynch, and M. Merritt, "Easy impossibility proofs for distributed consensus problems," in *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '85. New York, NY, USA: Association for Computing Machinery, 1985, p. 59–70. [Online]. Available: https://doi.org/10.1145/323596.323602
[22] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 656–666, 1983. [Online]. Available: https://doi.org/10.1137/0212045
[23] F. Sheer Hardwick, A. Gioulis, R. Naeem Akram, and K. Markantonakis, "E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1561–1567.
[24] Y.-X. Kho, S.-H. Heng, and J.-J. Chin, "A review of cryptographic electronic voting," *Symmetry*, vol. 14, no. 5, 2022. [Online]. Available: https://www.mdpi.com/2073-8994/14/5/858
[25] A. Antipa, D. Brown, R. Gallant, R. Lambert, R. Struik, and S. Vanstone, "Accelerated verification of ECDSA signatures," in *Selected Areas in Cryptography*. Springer, 2005, pp. 307–318.
[26] T. Pornin, "Optimized lattice basis reduction in dimension 2, and fast Schnorr and EdDSA signature verification," Cryptology ePrint Archive, Paper 2020/454, 2020.
[27] "Digital signature standard (initial public draft)," National Institute of Standards and Technology, Federal Information Processing Standards Publication 186-5, 2019. [Online]. Available: {https://csrc.nist.gov/pubs/fips/186-5/ipd}
[28] "Public comments received on draft FIPS 186-5: Digital signature standards (DSS)," National Institute of Standards and Technology, Tech. Rep., 2021. [Online]. Available: {https://csrc.nist.gov/files/pubs/fips/186-5/ipd/docs/fips-186-5-draft-comments-received.pdf}
[29] "ISO/IEC 9834-1:2012 - information technology — procedures for the operation of object identifier registration authorities — part 1: General procedures and top arcs of the international object identifier tree," International Organization for Standardization, Standard, 2012.
[30] J. Brendel, C. Cremers, D. Jackson, and M. Zhao, "The provable security of Ed25519: Theory and practice," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1659–1676.