IACR Transactions on Symmetric Cryptology
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–20. DOI:XXXXXXXX

# Faster signature verification with 3-dimensional decomposition

Vojtěch Suchánek[1], Marek Sýs[1] and Lukasz Chmielewski[1]

Masaryk University, Brno, Czech Republic

**Abstract.** We introduce a novel technique for verifying Schnorr signatures using fast endomorphisms. Traditionally, fast endomorphisms over prime field curves are used to decompose a scalar into two scalars of half of the size. This work shows that the context of the verification of signatures allows for the decomposition into three scalars of a third of the size. We apply our technique to three scenarios: verification of a single Schnorr signature, batch verification, and verification of BLS signatures within the Blind Diffie-Hellman key exchange protocol. Our experiments on AMD x86 and ARM Cortex-M4 platforms show performance improvements of approximately 20%, 13%, and 27%, respectively. The technique can also be used to accelerate the verification of ECDSA signatures, provided that one additional bit of information is appended to the signature.

As part of our work, we analyze the performance of 3-dimensional lattice reduction algorithms, which are critical for multi-scalar decompositions. To identify the most efficient approach, we experimentally compare Semaev's algorithm — known for its best asymptotic complexity — with the simpler Lagrange's algorithm. Our results reveal that, despite its simplicity, Lagrange's algorithm is nearly twice as fast as Semaev's in practice.

**Keywords:** elliptic curves · Schnorr · verification · GLV · endomorphisms

## 1 Introduction

Classical elliptic curve signatures, including ECDSA or Schnorr signature, are the key components of modern cryptographic systems. Their importance grew even further with their adoption into the Bitcoin cryptocurrency [Nak08], where each transaction contains a signature that is later verified as proof of ownership. In systems like Bitcoin, where nodes frequently process and validate large volumes of transactions, the efficiency of signature verification becomes a critical performance factor. The need for efficiency was one of the key motivations behind the introduction of Schnorr signatures to Bitcoin, as they offer improved verification speed compared to traditional ECDSA signatures, along with other benefits such as linearity and smaller signature sizes.

To verify a Schnorr signature[1], one needs to perform a (multi-)scalar multiplication on an elliptic curve to verify an equality $uG + vQ = R$, where $Q$ is the public key point, $G$ is a generator on the curve and the scalars $u, v$ with the point $R$ are derived from the signature. An equivalent equation is checked in the ECDSA verification, except only the $x$-coordinates of both sides are checked.

The literature contains numerous techniques to speed up scalar multiplication to compute $vQ$ or $uG + vQ$, in particular [HMV03; Was08]. Generally, the cost of these techniques is proportional to the sizes of the scalars $u$ and $v$, which motivated several

[1]More variants of the verification exist. We use the one usually considered in recent works, including the Bitcoin protocol [WNR20].

Licensed under Creative Commons License CC-BY 4.0.

methods to lower their size [GLV01; GLS09]. One of them is the GLV method [GLV01] that computes the scalar multiplication $vQ$ via the multi-scalar multiplication $v_0Q + v_1\varphi(Q)$ using an efficiently computable endomorphism $\varphi$. The scalars $v_0$ and $v_1$ have half the bit-length of $v$, which eliminates half of the doubling operations on the curve and slightly reduces the number of additions. Brumley [Bru15] reports 50% acceleration using the GLV curves making them the fastest elliptic curves in the OpenSSL library. Popular examples of curves with efficiently computable endomorphisms are the Bitcoin curve secp256k1 or many pairing-friendly curves, such as BLS12-381 [Bow17].

In principle, decomposing the scalar $v$ into more than two smaller scalars $(v_0, v_1, v_2, \dots)$ can lead to greater speed-ups [GLS09]. However, existing methods that support such multi-dimensional decompositions rely on elliptic curves defined over extension fields. As a result, prime field curves have, so far, been limited to 2-dimensional decompositions. This work introduces a 3-dimensional decomposition for the verification of signatures over prime field curves. More precisely, we present the following **contributions:**

- We present a new technique, called GLV-3, for the verification $uG + vQ = R$ using a fast endomorphism that applies to all Schnorr-like signatures, including modified ECDSA signatures. We decompose the original scalars $u, v$ into six third-size scalars resulting in a 20% speed-up for the verification of single signatures and 13% speed-up for batch verification. Furthermore, we also provide arguments for why larger scalar decompositions (to four scalars) may be suboptimal.

- We apply the technique to the Blind Diffie-Hellman key exchange protocol [Wag96], gaining a 25% speed-up for signature verification.

- The result are experimentally demonstrated on a 64-bit AMD x86 processor and on a 32-bit ARM Cortex-M4 STM32F407 microcontroller over 384-bit and 521-bit curves (see Table 1). This demonstrates that the approach applies not only to modern CPUs, which are commonly used in server applications but also to embedded architectures found in secure elements, such as smartcards[2].

- To select an efficient 3-dimensional lattice reduction algorithm for the decomposition of scalars, we experimentally compare Semaev's algorithm [Sem01] and the naive Lagrange's 3-dimensional algorithm. While Semaev's algorithm is asymptotically better than Lagrange's (quadratic vs cubic complexity), the results reveal that the latter is up to 45% faster on both the x86 processor and the microcontroller.

- We show that a 4-dimensional variant of our technique can be used to improve the complexity of the dependent coordinates problem (DCP) used in the zero-value point (ZVP) side-channel attacks [SCJ+21]. The ZVP attack can recover a static ECDH key but is limited by the underlying DCP problem. Our technique reduces the complexity of DCP from $O(n)$ to $O(\sqrt{n}\log^3 n)$.

**Outline.** In Section 2, we provide a background on elliptic curves, the GLV method, and the use of scalar decomposition in the verification of signatures. Section 3 contains the contributions of our work. In Section 3.1, we experimentally compare lattice reduction algorithms and select the fastest one for our 3-dimensional decomposition. In Section 3.2, Section 3.3 and Section 3.4 we design and experimentally demonstrate our GLV-3 technique on the verification in BDHKE, verification of Schnorr signatures and batch verification, respectively. Section 4 applies our technique to improving the Zero-value Point attack. Finally, we conclude in Section 5.

[2]The source code is available at https://anonymous.4open.science/r/glv3-55DF.

Table 1: The speed-up of our 3-dimensional GLV technique when compared to the state-of-the-art approach for three different verification algorithms.

| | bits | AMD x86 | ARM Cortex-M4 |
|---|---|---|---|
| Single Schnorr | 384 | 18% | 18% |
| | 521 | 19% | 20% |
| Batch Schnorr | 384 | 11% | 11% |
| | 521 | 13% | 13% |
| BDHKE | 384 | 23% | 25% |
| | 521 | 24% | 27% |

# 2 Background

We focus on elliptic curves in the short Weierstrass form $E/\mathbb{F}_p : y^2z = x^3 + axz^2 + bz^3$, where $a, b \in \mathbb{F}_p$ and $p \geq 5$ is a prime. The set of projective points $(x : y : z)$ over $\mathbb{F}_p$, which forms a finite abelian group with a neutral point $\infty = (0 : 1 : 0)$, is denoted $E(\mathbb{F}_p)$. As usual in elliptic curve cryptography, we assume that there is a large subgroup of prime order $n \approx p$ and $h$ is the cofactor, i.e., $\#E(\mathbb{F}_p) = nh$. Denote $t$ the bit-length of $n$.

For any point $P \in E(\mathbb{F}_p)$, denote its coordinates by $x_P, y_P, z_P$. Apart from these standard projective coordinates, we also use the affine coordinates $(x_P, y_P) = (x_P : y_P : 1)$ and Jacobian coordinates [BL07]. The addition of two points $P_1 + P_2$ can be defined using polynomial functions in their coordinates $x_{P_i}, y_{P_i}, z_{P_i}$ [Was08]. The case $P_1 = P_2$ is often treated separately using a more efficient doubling formula, though this depends on the used coordinate system.

We can define scalar multiplication $kP$ as the $k$-fold addition $P + \cdots + P$. This can be effectively computed using several algorithms that generally iterate over the digits of $k$ in an appropriate representation, computing point doublings and additions in each iteration. A standard example is the double-and-add algorithm (Algorithm 1), which performs, on average, $t$ doublings, one for each bit of $k$, and $t/2$ additions, one for each bit equal to 1.

**Algorithm 1:** double-and-add

**Input:** Point $P$, $t$-bit scalar $k = (k_{t-1}, k_{t-2}, \ldots, k_0)$
**Output:** $kP$

```
1 Q ← ∞
2 for i = t − 1 down to 0 do
3 |   Q ← 2Q
4 |   if k_i = 1 then Q ← Q + P;
5 return Q
```

## 2.1 GLV method

Gallant et al. [GLV01] introduced a technique to accelerate scalar multiplication using *fast endomorphisms* — the GLV method. An endomorphism $\varphi : E \to E$ is a group homomorphism defined by rational maps. Any such endomorphism naturally acts as a scalar multiplication by $\lambda_\varphi \in \mathbb{Z}$ on the (cyclic) subgroup of prime order $n$. Any scalar $k \in \mathbb{Z}$ can be then decomposed using the extended Euclidean algorithm as

$$k = k_0 + k_1\lambda_\varphi \pmod{n}, \quad k_0, k_1 \approx \sqrt{n}.$$

The benefit of this decomposition is that $k_0, k_1$ have half the bit-length of $k$, and so the number of iterations in the scalar multiplication is also halved if the Straus-Shamir trick

[BS64] is used: Instead of computing $kP$, we can compute $k_0P + k_1\varphi(P)$ by iterating through the bits of $k_0, k_1$ at the same time. In each iteration, we add $P$, $\varphi(P)$, or $P+\varphi(P)$ depending on the bits. The points $\varphi(P)$, $P + \varphi(P)$ are precomputed at the beginning. The cost of this multiscalar multiplication is roughly $t/2$ doublings and $3t/8$ additions — half the amount of doublings and $t/8$ less additions than the double-and-add.

The joint sparse form (JSF) [Sol01; Pro03] can be used to reduce the number of additions. In general, JSF is a joint representation of two or more scalars that represents each pair of bits (one bit for each scalar) using $0, 1$ or $-1$ such that the number of non-zero pairs is minimal. The joint Hamming weight (the number of non-zero pairs) then equals the number of additions during the Straus-Shamir trick. The size of the JSF form is larger than the binary form by at most one (signed) bit and so the number of additions only increases by at most one. The expected joint Hamming weight of two $s$-bit scalars in JSF is $0.5s$, whereas, in simple binary, it would be $0.75s$. We will also consider the JSF of three and four scalars for which the expected weight is $0.59s$ and $0.64s$, respectively.

In the case of GLV, the two $t/2$-bit scalars $k_0, k_1$ have joint Hamming weight $t/4$ resulting in $t/2$ doublings and $t/4$ additions (Algorithm 2). For comparison, the simple double-and-add with the non-adjacent form (NAF) representation [HMV03] performs $t$ doublings and $t/3$ additions, i.e., twice as many doublings and $t/12$ more additions.

**Algorithm 2:** GLV method with Straus-Shamir trick

**Input:** Point $P$, $t/2$-bit scalars $k_0$, $k_1$, endomorphism $\varphi$
**Output:** $k_0P + k_1\varphi(P)$

```
T[j0, j1] ← j0P + j1φ(P) for all (j0, j1) ∈ {0, 1, −1}^2
Q ← ∞
for (j0, j1) ∈ JSF(k0, k1) do
    Q ← 2Q
    Q ← Q + T[j0, j1]
return Q
```

Naturally, the GLV method makes sense only if $\varphi(P)$ can be much more efficiently computed than the scalar multiplication by $\lambda_\varphi$. Elliptic curves in the form $y^2 = x^3 + b$ (i.e., $a = 0$), where $p = 1 \pmod 3$, are equipped with an endomorphism which can be computed using only one multiplication in $\mathbb{F}_p$. More precisely, they have an endomorphism $\varphi : (x, y) \mapsto (\beta x, y)$, where $\beta \in \mathbb{F}_p^\times$. The corresponding scalar $\lambda_\varphi$ satisfies the quadratic equation $\lambda_\varphi^2 + \lambda_\varphi + 1 = 0 \pmod n$. Popular examples of such curves are the secp256k1 curve used in Bitcoin and the pairing-friendly curve BLS12-381 [Bow17].

## 2.2 Scalar decomposition

Algorithm 2 takes on input the decomposed scalar $k = k_0 + k_1\lambda_\varphi \pmod n$. The usual use case (e.g., ECDSA) assumes that $k$ is not fixed and so the scalar decomposition needs to be done efficiently on the fly. The common approach is to consider the lattice

$$\mathcal{L} = \langle (z_0, z_1) \mid z_0 + z_1\lambda_\varphi = 0 \pmod n \rangle$$

and find a close vector $v \in \mathcal{L}$ to the vector $(k, 0)$. The decomposition $(k_0, k_1)$ is the short vector $(k, 0) - v$. Assuming we have a short basis of $\mathcal{L}$, with vector norms $\approx n^{1/2}$, Babai's technique finds $v$ resulting in $k_0, k_1 \approx n^{1/2}$ [Bab86].

The bottleneck is finding the short basis for $\mathcal{L}$, but this can be done once for the curve using the following early abort version of the extended Euclidean algorithm (EEA): Recall that for any two integers, $(n, \lambda_\varphi)$ in our case, EEA produces a sequence of equations

$$s_i n + t_i \lambda_\varphi = r_i, \quad \text{where } s_0 = 1,\ t_0 = 0,\ r_0 = n,\ s_1 = 0,\ t_1 = 1,\ r_1 = \lambda_\varphi,\ r_i \geq 0.$$

The sequence $r_i$ is strictly decreasing, starting from $n$ until it reaches the greatest common divisor of $\lambda_\varphi$ and $n$. The sequence $|t_i|$ is strictly increasing, starting from 0. Heuristically, the sequences are expected to meet in the middle when $r_i \approx |t_i| \approx n^{1/2}$, in which case the algorithm is aborted, and two smallest vectors $(r_i, -t_i)$ are taken as the basis of $\mathcal{L}$. This does not work in general. For example, when $\lambda_\varphi < n^{1/2}$ then no short basis is found since the algorithm terminates at $i = 1$ with $(\lambda_\varphi, -1)$, $(n, 0)$.

Since one endomorphism enables us to decompose $k$ into two smaller scalars, one would like to use more endomorphisms for larger decompositions. This is the idea of the GLS method [GLS09], which uses curves over extension fields that are equipped with two fast endomorphisms $\varphi$ and $\psi$:

$$k = k_0 + k_1\lambda_\varphi + k_2\lambda_\psi + k_3\lambda_\varphi\lambda_\psi \pmod n, \quad k_i \approx \sqrt[4]{n}.$$

Similarly to the 2-dimensional case, the decomposition $(k_0, k_1, k_2, k_3)$ is the vector $(k, 0, 0, 0) - v$, where $v$ is the closest vector to $(k, 0, 0, 0)$ in the lattice

$$\mathcal{L} = \langle (z_0, z_1, z_2, z_3) \mid z_0 + z_1\lambda_\varphi + z_2\lambda_\psi + z_3\lambda_\varphi\lambda_\psi = 0 \pmod n \rangle.$$

The part that does not generalize easily is the search for the short basis of $\mathcal{L}$. Usually, the short basis is found by applying some reduction algorithm (e.g., LLL [LLL82]) to the (long) basis: $(\lambda_\varphi, -1, 0, 0), (\lambda_\psi, 0, -1, 0), (\lambda_\varphi\lambda_\psi, 0, 0, -1), (n, 0, 0, 0)$. The short basis can be precomputed again once for the given curve, so this does not have any effect on the performance. The main limitation of the GLS method is that it is only supported by curves over extension fields. Our work focuses on prime field curves that are used in cryptographic protocols.

## 2.3 Signatures

This part summarizes the signature schemes considered in this paper. Let $E$ be an elliptic curve as defined above. Denote $G$, the generator of the prime subgroup of order $n$, and $H$, a cryptographic hash function. For all the signature schemes denote $d \in [1, n]$ the private key, $P = dG$ the public key, and $m$ any message.

**Schnorr signature.** Schnorr signature over $E$ is a pair $(R, s)$, where $R = rG$ for a random $r \in [1, n]$ and $s = r + H(R, m)d \pmod n$ Here, $R$ is encoded using only $x_R$ for the hash function. To verify $(R, s)$ anyone can then check the equation $sG = R + H(R, m)P$. To save space, the signature $(R, s)$ can contain only the coordinate $x_R$ and one additional bit signaling the parity of $y_R$, which is then used for the recovery of $y_R = \sqrt{x_R^3 + ax_R + b}$ (another option is to force the bit during the generation of $R = rG$). Note that there is another variant of the Schnorr signature defined as $(e, s)$, where $e = H(sG - eP, m)$. We only focus on the first, and more popular variant that supports batch verification and is part of the Bitcoin improvement proposal (BIP) 340 [WNR20].

**Modified ECDSA.** The classical ECDSA signature is the pair $(r, s)$, where $r = x_{kG} \pmod n$ for a random $k \in [1, n]$ and $s = k^{-1}(H(m) + rd) \pmod n$. Anyone can then verify $(r, s)$ by computing $R = H(m)s^{-1}G + rs^{-1}P$ and checking that $x_R \pmod n = r$. ECDSA signatures can be verified in a similar way as Schnorr signatures, i.e., checking that $R = H(m)s^{-1}G + rs^{-1}P$, assuming we have the point $R$ (and not merely $r$). Antipa et al. [ABG+05] considered two versions of ECDSA which include the coordinates of $R$ in the signature or append enough information for the reconstruction of the coordinates. For prime order curves, including the popular NIST curves or secp256k1, one bit is enough. This work will focus on these Schnorr-like versions of ECDSA (denoted ECDSA*) as well.

**BLS signature.** The BLS signature [BLS01] is the point $\sigma = dH_E(m)$, where $H_E$ is an appropriate hash-to-curve function, i.e., $H_E(m)$ is a point on $E$. The verification equation

is $e(\sigma, G) = e(H_E(m), P)$. Here, $e$ is a bilinear pairing. Hence, only pairing-friendly curves (with efficiently computable pairing) can be used in practice. While the signing itself is simple and fast, the computation of the pairing can be a limiting factor, especially for resource-constrained devices.

**BDHKE.** The Blind Diffie-Hellman Key Exchange (BDHKE) protocol [Wag96] is not a classical signature algorithm, but uses the idea of the BLS signature and serves as another target of our work. The goal of this protocol is to make the signer sign a blinded value for a second party (user) so that when the unblinded value is revealed, the signer can verify it but cannot trace it back to the user. The exact steps of the protocol between a user and a signer are the following:

- User computes the point $h(m)$ as in the BLS signature and blinds it by adding a factor $rG$ for a random $r$: $C = H_E(m) + rG$. The user sends $C$ to the signer.
- The signer computes $T' = dC$ and sends it back to the user.
- The user unblinds the point $T'$ using the signer's public key: $T = T' - rP$.
- Later, when the signer is provided $(m, T)$, they verify that $dH_E(m) = T$.

The protocol has been used for Cashu – an e-cash system backed by Bitcoin [Ope]. One of the key properties of this protocol is that the verification is done only by the owner of the private key. This means that the costly pairing-based verification can be replaced by a simple signature reconstruction.

**Remark 1.** The verification equation in the popular EdDSA signature scheme [BDL+12] is the same as in the Schnorr signature. Yet, we exclude it from this work for the following reasons. EdDSA is designed for Edwards curves with complete addition formulas. More precisely, EdDSA uses a curve of the form $-x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{F}_p$, where $d \in \mathbb{F}_p$ is a non-square and $p = 1 \pmod 4$ (implying that $-1$ is a square in $\mathbb{F}_p$). Our work uses GLV curves that have an efficient endomorphism of degree 1 that are of the form $y^2 = x^3 + b$ or $y^2 = x^3 + ax$; these curves are precisely the ones with endomorphism ring with nontrivial units that correspond to endomorphisms with their rational maps of degree 1. Such GLV curves cannot be transformed to the Edwards curve. The reason follows from the birational equivalence between Weierstrass curves $y^2 = x^3 + ax + b$ and twisted Edwards curves [BBJ+08]. It can be shown that the Edwards $d$ coefficient must be equal to $3\alpha^2 + a$ for the equivalence to exist, where $\alpha$ is a root of $x^3 + ax + b$. For both considered forms of GLV curves, simple expression manipulation shows that $d$ is always a square, breaking the EdDSA assumption. Nevertheless, curves with less efficient endomorphisms (of higher degree) could be considered, but we leave this as a future work.

## 2.4 Verification of signatures

The focus of this work is the verification of Schnorr signatures (VSS), ECDSA* signatures, and the verification of scalar multiplication (VSM) in BDHKE. These can be generalized by the following two problems:

*(VSM)* Given two affine points $P, R$ and a scalar $k \in \mathbb{Z}$, decide:

$$kP \stackrel{?}{=} R.$$

*(VSS)* Let $G$ be a fixed generator. Given two affine points $Q, R$ and two scalars $u, v \in \mathbb{Z}$, decide:

$$uG + vQ \stackrel{?}{=} R.$$

The first and obvious approach to both problems is to recompute $kP$ or $uG + vQ$ and compare with the other side of the equation — the point $R$. The ECC literature contains a plethora of techniques for (multi-)scalar multiplication [HMV03; CFA+12]. The overall principle is to iterate over the digits or windows of the scalars in some representation. Depending on the context, the techniques might assume that one of the points is fixed and leverage some precomputation. Additionally, the threat of side-channel attacks is also often considered in the literature.

In our context, we assume that only the point $G$ is fixed in VSS. Furthermore, we do not consider side-channel attacks, as only public data are verified. Our work follows the observation that the cost of (multi-)scalar multiplication is proportional to the bit-lengths of the involved scalars. Two notable techniques that are based on this idea can be summarized in the following way.

First, the GLV method (Section 2.5) decomposes the scalar $k$ (or $v$ for VSS) into $k = k_0 + k_1\lambda_\varphi \pmod n$ using a scalar $\lambda_\varphi$ and then applies Straus-Shamir trick to check whether $k_0P + k_1\lambda_\varphi P = R$. It works with the assumption that $\lambda_\varphi P$ can be efficiently computed as $\varphi(P)$.

$$kP \stackrel{?}{=} R \quad \rightarrow \quad k = k_0 + k_1\lambda_\varphi \pmod n \quad \rightarrow \quad k_0P + k_1\varphi(P) \stackrel{?}{=} R \qquad \text{(GLV)}$$

The second method by Antipa et al. [ABG+05] (Section 2.6) similarly splits the scalar $k$ by decomposing 0 into $0 = k_0 - k_1k \pmod n$ using the scalar $k$. Then it applies the Straus-Shamir trick to check $k_0P - k_1kP = \infty$. The assumption here is that the equation to be verified $kP = R$ holds and $kP$ can be efficiently 'computed' by simply taking $kP = R$.

$$kP \stackrel{?}{=} R \quad \rightarrow \quad 0 = k_0 - k_1k \pmod n \quad \rightarrow \quad k_0P - k_1R \stackrel{?}{=} \infty \qquad \text{([ABG+05])}$$

We will explain the full details of the above ideas in Section 2.5 and Section 2.6. To express the cost of the described algorithms, we use the following notation: $A$ and $D$ denote the cost of point addition and doubling, respectively. Also, recall that we work with points of order $n$, where $n$ has $t$ bits.

## 2.5 Verification with GLV

Section 2.1 described the GLV method that can be used for VSM (Algorithm 2). We will now compute the cost in terms of the number of point additions $A$ and doublings $D$, as well the number of field multiplications $M$, i.e., the cost in the form $c_1A + c_2D + c_3M$. We will also use the standard Jacobian formulas [BL07] with $A = 11M$, $D = 7M$, to express the cost purely in the number of multiplications, i.e. $cM$. We have selected these formulas as they are generally the fastest for Weierstrass curves with $a = 0$ (in terms of the number of field multiplications), and they are also implemented in the Mbed TLS library that we used for experiments. Furthermore, we assume that the curve admits an endomorphism $\varphi$ with the cost of its application equal to one finite field multiplication.

For the precomputation in Step 1 of Algorithm 2, it is helpful to use the fact that opposite points differ only by a sign in the $y$-coordinate and so we can compute and store only half the points. This amounts to the input point $P$ and three points $\varphi(P)$, $P+\varphi(P) = -\varphi(\varphi(P))$, $P-\varphi(P)$ that can be computed with the cost $M$, $M$, $A$, respectively. The evaluation part uses $t/2$ doublings. The number of additions is given by the joint Hamming weight of the two $t/2$-bit scalars in JSF, which is $\frac{t}{4}$ on average. The total cost is then

$$\left(\frac{t}{4} + 1\right) A + \frac{t}{2}D + 2M = (6.25t + 13)M.$$

The idea can be adaptedto the VSS problem (as in the libsecp256k1 library [Wui13]) by decomposing both the scalars $u, v$ and computing $u_0G + u_1\varphi(G) + v_0Q + v_1\varphi(Q)$ using

two simultaneous Straus-Shamir tricks. We precompute the points for $u_0G + u_1\varphi(G)$ and $v_0 + v_1\varphi(Q)$ separately and then run the evaluation at the same time. The combined Hamming weight of the two pairs of scalars in JSF is $2\frac{t}{4} = \frac{t}{2}$. The overall cost is then

$$\left(\frac{t}{2} + 2\right) A + \frac{t}{2}D + 4M = (9t + 26)M.$$

## 2.6 Verification by Antipa et al.

Antipa et al. [ABG+05] do not apply the simple recomputation of the (multi-)scalar multiplication to the VSS and VSM problems. Instead of using the point $R$ just for the final comparison, they use it for the computation of the scalar multiplication. The idea is to express $k$ as a fraction $k = \frac{k_0}{k_1}$ of two smaller scalars of half the bit-length of $k$. Or from another point of view, we decompose 0 as $0 = k_0 - k_1 k$ using $k$. The pair $(k_0, k_1)$ is a short vector in the lattice

$$\langle (z_0, z_1) \mid z_0 - z_1 k = 0 \pmod{n} \rangle$$

and can be found using EEA as described in Section 2.2. To solve VSM, we then use the Straus-Shamir trick to compute $k_0P - k_1R$ and check whether the result is $\infty$.

Applying this method to VSS is a bit more tricky and needs to use the fact that $G$ is fixed: We can decompose $0 = v_0 - v_1 v \pmod{n}$, $v_0, v_1 \approx n^{1/2}$, which gets us $uv_1G + v_0Q - v_1R = \infty$. The only scalar bigger than $n^{1/2}$ is $uv_1$, but we can use the fact that $G$ is a fixed point and the multiple $G' = 2^{t/2}G$ can be precomputed once and stored. Using $G'$, we can split $uv_1$ into the upper and lower binary halves $u_U, v_L$ which gets us the following verification equation:

$$u_LG + u_UG' + v_0Q - v_1R = \infty.$$

The left side is then computed as in Section 2.5 with a similar overall cost (slightly worse since the endomorphism cannot be used in the precomputation). Antipa et al. [ABG+05] report a speed-up of 36% on the ARM7TDMI platform when compared to the simple Straus-Shamir trick on $uG + vQ$.

Compared to the GLV approach (Section 2.5), this method has the advantage that no endomorphisms are necessary. One could get the idea of using both methods to reduce the size of the scalars even further. Unfortunately, they are not compatible in any trivial way. For instance, we could try to decompose $v = v_0 + v_1\lambda_\varphi$ using $\lambda_\varphi$ and then try to express $v_0, v_1$ as fractions of smaller scalars. The problem is that since already $v_i \approx n^{1/2}$, they are not likely to be fractions of much smaller scalars. The EEA would most likely give us trivial equality $v_0 = \frac{v_0}{1}$. Trying the opposite way (decomposition and then fractions) suffers from the same problem. The decompositions of $v_0, v_1$ using $\lambda_\varphi$ for $v = \frac{v_0}{v_1}$ are most likely to be trivial as well. Hence, a simple application of one method after the other does not work.

Antipa et al. [ABG+05] extended their idea of 4 half-size scalar multiplications to 6 third-size scalar multiplications using unbalanced fractions. The method discussed so far aborts EEA when $v_0$ (and consequently $v_1$) is $\approx n^{1/2}$. They show that in general, if one aborts at $v_0 \approx n^e$ then $v_1 \approx n^{1-e}$. Selecting $e = 2/3$, we get $uv_1G + v_0Q - v_1R = \infty$, where $uv_1 \approx n$, $v_0 \approx n^{2/3}$ and $v_1 \approx n^{1/3}$. If we assume that the multiple $Q' = 2^{2t/3}Q$ is available, then we can split $v_0$ into the upper and lower binary $v_U, v_L$ halves of size $n^{1/3}$. Similarly, we split $uv_1$ into three scalars $u_U, u_M, u_L \approx n^{1/3}$. All together, this gets us

$$u_LG + u_MG' + u_UG'' + v_UQ + v_LQ' - v_1R = \infty,$$

where $G' = 2^{t/3}G$, $G'' = 2^{2t/3}G$ are precomputed once. Using three Straus-Shamir tricks with JSF, the cost is $t/2$ additions and $t/3$ doublings [ABG+05]. The main limitation of

this method is that the multiple $Q'$ must be precomputed and available for verification. One option is to append it to the signature, and another is to include it in the public key certificate as it depends on the public key only. Either way, this cannot be used for the current signature schemes, and so we will not discuss this further. In the following section, we present a new method with 6 third-size scalar multiplications without this assumption.

# 3 Verification using 3-dimensional GLV

This section presents the contribution of our work: a 3-dimensional GLV method that accelerates the verification of signatures (VSS) and scalar multiplication (VSM) using a fast endomorphism. The section contains theoretical analysis and results of experiments.

The two presented approaches (Section 2.5 and Section 2.6) to VSM and VSS reduced the size of the involved scalars using an appropriate decomposition. The GLV method decomposes $k = k_0 + k_1\lambda_\varphi \pmod n$ using $\lambda_\varphi$, with the assumption that we have an efficient endomorphism such that $\varphi(P) = \lambda_\varphi P$. The second method by Antipa et al. [ABG+05] decomposes $0 = k_0 - k_1 k \pmod n$ using $k$, assuming that the equality $kP = R$ is true.

Our method (that we call *3-dimensional GLV* or GLV-3 for short) combines both of the methods described above and considers the decomposition of 0 by $k$ and $\lambda_\varphi$ at the same time: $0 = l_0 + l_1\lambda_\varphi + l_2 k \pmod n$. It essentially imitates the GLS method that uses two endomorphisms for a decomposition by taking $\lambda_\varphi$ and $k$ as the endomorphisms. For VSM, we verify the following equation:

$$(l_0 + l_1\lambda_\varphi + l_2 k)P = l_0 P + l_1\varphi(P) + l_2 R = \infty,$$

*assuming* the equality $kP = R$ is true. To find an appropriate triple $(l_0, l_1, l_2)$, we follow the GLS method and consider the lattice generated by the vectors $(k, 0, -1), (\lambda_\varphi, -1, 0)$, and $(n, 0, 0)$. Assuming $k$ is a large random scalar, which is the case for signature schemes specified above, we should expect a short vector $(l_0, l_1, l_2)$ in the lattice of norm $\approx n^{1/3}$.

In Section 3.1, we select an appropriate lattice reduction technique for finding the short vector $(l_0, l_1, l_2)$. Section 3.2 and Section 3.3 apply GLV-3 to VSM and VSS in full detail and present experimental results. Section 3.4 presents how the technique can be easily modified for batch signatures.

**Remark 2.** All of our algorithms were implemented in the Mbed TLS library [Fir24] and tested on a 64-bit AMD x86 processor and on a 32-bit ARM Cortex-M4 STM32F407 microcontroller (MC). We ran each experiment 1000 times on the AMD processor and 10 times on the MC [3]. We used elliptic curves over three different bit-lengths: 256, 384, and 521. For the 256-bit case, we used the popular `secp256k1` curve $y = x^3 + 7$ over $\mathbb{F}_p$ with $p = 2^{256} - 2^{32} - 977$. For the remaining two bit-lengths, we generated two curves of the same form $y^2 = x^3 + 7$ as `secp256k1` for easier implementation. The base field $\mathbb{F}_p$ is of the size $p = 2^{384} - 170937$ and $p = 2^{521} - 503629$, respectively (the bit-length 521 was chosen based on the NIST P-521 curve). Finally, the experiments for batch verification were only simulated in Python, and the cost was approximated based on the previous experiments on both platforms. We have to emphasize that the goal of the experiments is not to break the records for signature verification on the platforms. Rather, our goal is a fair comparison between individual algorithms and so we are interested in the relative performance.

## 3.1 Reducing 3-dimensional lattices

This part describes our choice of a lattice reduction algorithm for the lattice given by the vectors

$$(k, 0, -1), (\lambda_\varphi, -1, 0), (n, 0, 0).$$

[3] This is a reasonable number since, for the MC, we measure execution time in clock cycles. We expect to obtain always the same number since our algorithms are not randomized.

Recall that the classical 2-dimensional GLV (or 4-dimensional GLS) decomposition precomputes the reduced basis (e.g., using the LLL algorithm) once and then uses it to decompose each scalar efficiently. The situation with the method by Antipa et al. [ABG+05] is slightly more complicated since the lattice is defined by the given scalar, and the reduction cannot be precomputed. However, the dimension is only 2, and the efficient EEA can be used, which makes the cost of the reduction negligible when compared to the scalar multiplication. Our GLV-3 method inherits the worst of both worlds as we have a 3-dimensional lattice defined per each scalar. Our goal is to accelerate signature verification, and so the choice of the reduction algorithm is crucial.

Our strategy is to follow the work by Semaev [Sem01] on the reduction of 3-dimensional lattices. However, rather than selecting the lattice reduction algorithm with the best asymptotical complexity that always finds a good basis, we take the experimental approach. We select the algorithm that experimentally seems to work best for the typical sizes of input values in our context.

**Algorithm 3:** Lagrange's lattice reduction

**Input:** Basis $b_1, b_2$ of a lattice $\mathcal{L} \subset \mathbb{R}^2$
**Output:** Basis $b_1, b_2$ of $\mathcal{L}$ satisfying $|b_1| \leq |b_2|$, $2|\langle b_1, b_2\rangle| < |b_1|^2$

1 **do**
2  **if** $|b_1| > |b_2|$ **then** swap $b_1, b_2$
3  $m \leftarrow \lfloor \langle b_1, b_2\rangle / |b_1|^2 \rceil$
4  $b_2 \leftarrow b_2 - m b_1$
5 **while** $m \neq 0$
6 **return** $b_1, b_2$

Semaev [Sem01] builds up on the basic reduction algorithm for a general 2-dimensional lattice — the Lagrange's algorithm (Algorithm 3). The idea of Lagrange's algorithm for given vectors $b_1, b_2$ is to iteratively reduce the bigger vector, say $b_2$, by subtracting a multiple of the other vector, i.e., replace $b_2$ with $b_2 - mb_1$. The optimal choice for $m$ is to approximate the orthogonal projection of $b_2$ to the orthogonal complement of $b_1$: $m = \lfloor \langle b_1, b_2\rangle / |b_1|^2 \rceil$. Here, $|\cdot|$ is the Euclidean norm and $\langle \cdot, \cdot \rangle$ is the scalar product. The output of the algorithm is a *reduced pair* $b'_1, b'_2$ satisfying $|b'_1| \leq |b'_2|$, $2|\langle b'_1, b'_2\rangle| < |b'_1|^2$, which we denote $(b'_1, b'_2) = G(b_1, b_2)$ as in [Sem01]. The reduced pair forms the so-called Minkowski reduced basis and always contains the shortest vector in the lattice. Note that the costly division in Step 3 in Algorithm 3 can be computed with low precision since the result is then rounded to the closest integer. As a further optimization, the scalar products $\langle b_i, b_j\rangle$ are computed once at the beginning and stored in the Gram matrix. With each replacement in Step 4, we only update the Gram matrix, store $m$, and reconstruct the final vectors at the end.

As Semaev [Sem01] further shows, Algorithm 3 can be naively extended to 3 dimensions by iteratively applying Algorithm 3 to every pair until we get a basis in which each pair is reduced (Algorithm 4). We call such basis *2-reduced*. As Semaev explains, one can easily construct basis vectors that are 2-reduced but do not contain the shortest vector, and thus the algorithm is not guaranteed to find it. Luckily, this can be solved by appending one quick computation at the end of Algorithm 4 (see Algorithm 2 in [Sem01]). The algorithm then outputs the Minkowski reduced basis containing the shortest vector. Regarding asymptotic complexity, Algorithm 4 can take as many $O(\log^3 M)$ operations to find the 2-reduced basis, where $|b_1|, |b_2|, |b_3| \leq M$. As an example supporting this lower bound on the complexity, Semaev presents a specially constructed lattice with vectors $(1, 1, 0), (M, 0, 0), (0, M^2, 1)$ with significantly different sizes. However, as we will see, this is not the typical case.

Finally, Semaev presents an algorithm that finds the Minkowski reduced basis with

**Algorithm 4:** Lagrange's 3-dimensional lattice reduction

**Input:** Basis $b_1, b_2, b_3$ of a lattice $\mathcal{L} \subset \mathbb{R}^3$
**Output:** A 2-reduced basis $b_1, b_2, b_3$ of $\mathcal{L}$

**do**
  2-reduced $\leftarrow$ True
  **foreach** $i, j \in \{1, 2, 3\}$ **do**
    **if** $b_i, b_j$ *not* reduced **then**
      $b_i, b_j \leftarrow G(b_i, b_j)$ `// Using Algorithm 3`
      2-reduced $\leftarrow$ False
**while** not 2-reduced
**return** $b_1, b_2$

quadratic asymptotic complexity (Algorithm 5). The idea is to minimize the quadratic form $|b_3 - x_1 b_1 - x_2 b_2|$ in $x_1, x_2$, where $|b_1| \leq |b_2| \leq |b_3|$ and $b_1, b_2$ is a reduced pair. The optimal values for $x_1, x_2$ are found in real numbers, and the four possible combinations of the nearest integers are then checked. This is done repeatedly until the biggest vector cannot be reduced.

**Algorithm 5:** Semaev's lattice reduction

**Input:** Basis $b_1, b_2, b_3$ of a lattice $\mathcal{L} \subset \mathbb{R}^3$, s.t. $|b_1| \leq |b_2| \leq |b_3|$
**Output:** A reduced basis $b_1, b_2, b_3$ of $\mathcal{L}$

**while** True **do**
  $b_1, b_2 \leftarrow G(b_1, b_2)$
  $D \leftarrow 1 - \frac{\langle b_1, b_2 \rangle}{|b_1|^2} \cdot \frac{\langle b_1, b_2 \rangle}{|b_2|^2}$
  $y_1 \leftarrow -\frac{1}{D} \cdot \left( \frac{\langle b_1, b_3 \rangle}{|b_1|^2} - \frac{\langle b_1, b_2 \rangle}{|b_1|^2} \cdot \frac{\langle b_2, b_3 \rangle}{|b_2|^2} \right)$
  $y_2 \leftarrow -\frac{1}{D} \cdot \left( \frac{\langle b_2, b_3 \rangle}{|b_2|^2} - \frac{\langle b_1, b_2 \rangle}{|b_2|^2} \cdot \frac{\langle b_1, b_3 \rangle}{|b_1|^2} \right)$
  Select integers $x_1 \in [y_1 - 1, y_1 + 1], x_2 \in [y_2 - 1, y_2 + 1]$ such that $|b_3 + x_2 b_2 + x_1 b_1|$ is minimal
  $b_3' \leftarrow b_3 + x_2 b_2 + x_1 b_1$
  **if** $|b_3'| \geq |b_3|$ **then return** $b_1, b_2, b_3$
  $b_3 \leftarrow b_3'$
  Order $b_1, b_2, b_3$ by their norms.

Table 2 presents our experimental results of Lagrange's 3-dimensional algorithm (Algorithm 4) and Semaev's algorithm (Algorithm 5). The column 'Bits' shows the average bit-length of the shortest found vector norm. While Lagrange's algorithm is theoretically not guaranteed to find the shortest vector, it found it every time except for 4 out of 3000 cases (across the three bit-lengths). In these 4 cases, the sizes of the norms of the shortest found vectors for Semaev's and Lagrange's algorithms differed only by at most one bit. Surprisingly, Lagrange's algorithm was roughly 45% faster than Semaev's on the AMD. It was also much faster on the microcontroller, though it depended on the bit-length: $35\%, 45\%, 21\%$ for the bit-lengths $256, 384, 521$, respectively. This is in contrast with the theoretical comparison of the asymptotical complexities. As a result, we select Lagrange's Algorithm 4 for lattice reductions in the following sections.

**Remark 3.** Experimentally, we have found that in Algorithm 3, it is enough to perform the division on only the most significant byte of the numerator $\langle b_1, b_2 \rangle$ and denominator

Table 2: Comparison of implementations of Lagrange's 3-dimensional Algorithm 4 and Semaev's Algorithm 5 on a 64-bit AMD x86 processor and on a 32-bit ARM Cortex-M4 STM32F407 microcontroller. The columns show the asymptotic complexity, bit-lengths of inputs $k$ and $n$, average bit-length of the shortest vector norm, the runtime in milliseconds (AMD), and cycle count on the MC.

| | Complexity | $\log n$ | Avg. bits | Time (AMD) | Cycles (MC) |
|---|---|---|---|---|---|
| Lagrange | $\Omega(\log^3 n)$ | 256 | 84.813 | 0.062ms | 937211 |
| | | 384 | 127.593 | 0.097ms | 1738715 |
| | | 521 | 173.215 | 0.139ms | 2782500 |
| Semaev | $O(\log^2 n)$ | 256 | 84.811 | 0.110ms | 1449521 |
| | | 384 | 127.593 | 0.176ms | 3077910 |
| | | 521 | 173.214 | 0.257ms | 3538717 |

$|b_1|^2$. Similarly, in Algorithm 5, we have found that $y_1, y_2$ can be efficiently computed as

$$y_1 = -\frac{\langle b_1, b_3\rangle - \langle b_1, b_2\rangle \cdot \langle b_2, b_3\rangle}{|b_1||b_2|^2 - \langle b_1, b_2\rangle^2}, \quad y_2 = -\frac{\langle b_2, b_3\rangle - \langle b_1, b_2\rangle \cdot \langle b_1, b_3\rangle}{|b_1||b_2|^2 - \langle b_1, b_2\rangle^2},$$

where only the two most significant bytes of each term $\langle b_i, b_j\rangle$, or $|b_i|^2$ is considered. This differs from the original approach in [Sem04], which we found to be slightly slower for the selected bit-lengths. Furthermore, since the second and third vector of the initial basis $(k, 0, -1), (\lambda_\varphi, -1, 0), (n, 0, 0)$ are fixed for the given curve, we compute the reduction $b_1, b_2 \leftarrow G((\lambda_\varphi, -1, 0), (n, 0, 0))$ using Algorithm 3 once and store $u_1, u_2$. Finally, at the beginning we also decompose $k$ into $k_0, k_1$ as described in Section 2.2 and replace $(k, 0, -1)$ with $u_3 = (k_0, k_1, -1)$. Hence, the input vectors to both algorithms are the three vectors $u_1, u_2$, and $u_3$.

## 3.2 Scalar multiplication verification with GLV-3

We assume that the curve admits an endomorphism $\varphi$ with the cost of its application equal to one finite field multiplication. This section explains how our GLV-3 can be used to solve VSM faster than the previous methods (Section 2.5 and Section 2.6). As explained at the beginning of this section, we can transform the equation $kP = R$ (VSM) to $l_0P + l_1\varphi(P) + l_2R = \infty$, where $l_i \approx n^{1/3}$ are found using the Lagrange's 3-dimensional algorithm.

To verify $l_0P + l_1\varphi(P) + l_2R = \infty$, we compute the left-hand side using the classical Straus-Shamir trick with JSF for three scalars. We iterate through the signed bits of $l_0, l_1, l_2$ and add one of the 26 points $j_0P + j_1\varphi(P) + j_2R$, $j_i \in \{\pm 1, 0\}$, which are precomputed at the beginning. If we leverage the fast sign switch of points, it is enough to precompute 13 points (including the input points) with the cost of $9A+2M$. The cost of the Straus-Shamir trick is then given by the number of additions and doublings. The number of doublings is equal to the maximum of the bit-lengths of $l_0, l_1, l_2$, which is $\approx \frac{t}{3}$, where $t$ is the bit-length of $n$. The number of additions is given by the compound Hamming weight of $l_0, l_1, l_2$ which is $\approx 0.59 \cdot \frac{t}{3} \approx \frac{t}{5}$ in the JSF form. The overall cost is then

$$\left(9 + \frac{t}{5}\right) A + \frac{t}{3} D + 2M \approx (4.53t + 101)M.$$

Compared to the GLV method Section 2.5 with $(6.25t + 13)M$ complexity, this is a $\approx 22 - 25\%$ speed-up for the typical values of $t$ (between 256 and 521).

We have to emphasize that this 25% speed-up only considers the complexity of the Straus-Shamir trick in terms of additions and doublings, and the actual speed-up will

Table 3: Comparison of the GLV method and our GLV-3 for solving VSM. The first column shows the theoretical cost in the number of point additions, doublings, and field multiplications. The third column contains equivalent cost purely in the number of field multiplications for common bit-lengths. The last two columns show the speed in milliseconds (on a 64-bit AMD x86) and the cycle count (32-bit ARM Cortex-M4 MC).

| | Cost | $t$ | Mult. | Time (AMD) | Cycles (MC) |
|---|---|---|---|---|---|
| GLV | $(\frac{t}{4}+1)A+\frac{t}{2}D+2M$ | 256 | 1613M | 0.50ms | 12 668 727 |
| | | 384 | 2413M | 4.6ms | 226 492 679 |
| | | 521 | 3269M | 10.67ms | 539 123 262 |
| GLV-3 | $(9+\frac{t}{5})A+\frac{t}{3}D+2M$ | 256 | 1261M | 0.48ms | 11 048 969 |
| | | 384 | 1841M | 3.56ms | 170 030 994 |
| | | 521 | 2461M | 8.13ms | 400 341 296 |

be lower. The computation usually also involves, for example, a transformation of the precomputed points into an affine form. More importantly, we did not include the cost of Lagrange's algorithm to find $l_0, l_1, l_2$, which is a significant burden that the GLV does not have. Rather than theoretically assess the complexities of the GLV and our GLV-3, we experimentally compare them.

Table 3 shows the comparison for the common curve bit-lengths and the timing on our two platforms. For 384 and 521-bit curves, we measured the speed-up to be $23-24\%$ on the AMD and $25-26\%$ on the MC. For the 256-bit curves, it is only 5% on the AMD and 13% on the MC. This is mainly because of Lagrange's algorithm, whose cost grows with the bit-length (see Table 2), but not as rapidly as with the whole scalar multiplication.

## 3.3 Signature verification with GLV-3

This section applies GLV-3 to the verification of signatures (VSS). Let $uG+vQ=R$ be the verification equation. Using Algorithm 4, we can find scalars $l_0, l_1, l_2 \approx n^{1/3}$ such that $l_0 + l_1\lambda_\varphi + l_2 v = 0 \pmod n$. We multiply the verification equation by $l_2$, and after the substitution, we get $l_2uG - l_0Q - l_1\varphi(Q) - l_2R = \infty$. Now we split the scalar $l_2u \pmod n$ into three scalars of a third of the bit-length, i.e., $l_2u = u_0 + 2^{t/3}u_1 + 2^{2t/3}u_2 \pmod n$. Denote $G_1 = 2^{t/3}G$, $G_2 = 2^{2t/3}G$. Putting this all together, we get:

$$u_0G_0 + u_1G_1 + u_2G_2 - l_0Q - l_1\varphi(Q) - l_2R = \infty.$$

All of the scalars have $\approx t/3$ bits. To evaluate the left-hand side, we run two simultaneous Straus-Shamir tricks for $u_0, u_1, u_2$ and for $l_0, l_1, l_2$, both with JSF (Algorithm 6). More precisely, at the beginning, we precompute two sets of points: $h_0G_0 + h_1G_1 + h_2G_2$ and $j_0Q + j_1\varphi(Q) + j_2R$, $h_i, j_i \in \{\pm 1, 0\}$, where we exclude points with opposite signs. This amounts to 26 points computed with the cost $19A + 2M$. During the Straus-Shamir tricks, we iterate through the $t/3$ bits of the scalars $u_i, l_i$ and make two additions, one for each precomputed set. Since the compound Hamming weight of all the scalars is $2 \cdot 0.59 \cdot \frac{t}{3} \approx \frac{2t}{5}$, the total cost is

$$\left(19 + \frac{2t}{5}\right) A + \frac{t}{3}D + 2M \approx (6.73t + 211)M.$$

Compared to the GLV method with the cost $(9t+26)M$, this is a $17-21\%$ speed-up for $t$ between 256 and 521 bits.

Table 4 presents the experiment results. For 256 bits, the overhead of Lagrange's algorithm and the precomputation table is too large, and GLV-3 is only 2% faster than GLV on the MC and even 2% worse on the AMD. However, for 384 and 521 bits, the speed-up is $18-20\%$ on both platforms.

**Algorithm 6:** Signature verification using GLV-3

**Input:** $u, v \in \mathbb{Z}$, points $G$, $Q$, $R$, $G_1 = 2^{t/3}G$, $G_2 = 2^{2t/3}G$ on a curve with order $n$ and endomorphism $\varphi$

**Output:** $uG + vQ \stackrel{?}{=} R$

$l_0, l_1, l_2 \leftarrow$ Algorithm 4 on $(v, 0, -1), (\lambda_\varphi, -1, 0), (n, 0, 0)$

Find $u_0, u_1, u_2$, s.t. $ul_2 \pmod n = u_2|u_1|u_0$, where $|$ is conc. of bits.

Compute $h_0G + h_1G_1 + h_2G_2$ and $j_0Q + j_1\varphi(Q) + j_2R$ for $j_i, h_i \in \{\pm 1, 0\}$

$T \leftarrow \infty$

**for** $(h_0, h_1, h_2), (j_0, j_1, j_2) \in \mathrm{JSF}(u_0, u_1, u_2), \mathrm{JSF}(l_0, l_1, l_2)$ **do**

  $T \leftarrow 2T$

  $T \leftarrow T + h_0G + h_1G_1 + h_2G_2$

  $T \leftarrow T - j_0Q - j_1\varphi(Q) - j_2R$

**return** $T \stackrel{?}{=} \infty$

Table 4: Comparison of the GLV method and our GLV-3 for solving VSS. The first column shows the theoretical cost in the number of point additions, doublings, and field multiplications. The third column contains equivalent cost purely in the number of field multiplications for common bit-lengths. The last two columns show the speed in milliseconds (on a 64-bit AMD x86) and the cycle count (32-bit ARM Cortex-M4 MC).

| | Cost | $t$ | Mult. | Time (AMD) | Cycles (MC) |
|---|---|---|---|---|---|
| GLV | $(\frac{t}{2}+2)A + \frac{t}{2}D + 4M$ | 256 | 2330M | 0.81ms | 19 341 897 |
| | | 384 | 3482M | 7.13ms | 328 380 849 |
| | | 521 | 4715M | 15.72ms | 772 028 416 |
| GLV-3 | $\frac{t}{3}D + (19 + \frac{2t}{5})A + 2M$ | 256 | 1934M | 0.82ms | 18 902 318 |
| | | 384 | 2795M | 5.79ms | 268 629 338 |
| | | 521 | 3717M | 12.80ms | 615 912 490 |

## 3.4 Batch verification

Previous sections discussed accelerations of the verification of a single signature. Systems often need to verify a *batch* of signatures, so we will now modify the GLV-3 method for batch verification. The signatures in a batch might all correspond to one public key $Q$ or, more generally, to several $Q_i$. We will at first assume that the public key is fixed and focus on the general case afterward. The usual approach for batch signatures, proposed in [NMV+95] and since then adopted for EdDSA [BDL+12] and Schnorr signatures [WNR20], is to verify a random linear combination of the verification equations. Let $u_iG + v_iQ = R_i$ be the set of $B$ equations. The idea is to multiply each equation by a random integer $z_i$ and verify their sum:

$$\left(\sum_{i=1}^{B} z_i u_i\right) G + \left(\sum_{i=1}^{B} z_i v_i\right) Q - \sum_{i=1}^{B} z_i R_i = \infty.$$

The left side, containing $B + 2$ scalars and points, is then computed and checked with the neutral point on the right side. One of the state-of-the-art multiplication methods for multi-scalar multiplication, also used for batch signatures in [BDL+12], is the Bos-Coster method [De 94]. To compute the general expression $k_1P_1 + k_2P_2 + \cdots + k_sP_s$, where $k_1 \geq k_2 \geq \cdots \geq k_s$, the Bos-Coster method recursively computes $(k_1 - k_2)P_1 + k_2(P_1 + P_2) + k_3P_3 \cdots + k_sP_s$.

The complexity of the Bost-Coster method is determined by the number of iterations, which equals the number of point additions. In general, it is difficult to find a closed formula for the number of iterations. Hence, we simulated the execution for the usually

Table 5: The cost of the GLV method and our 3-dimensional GLV for batch verification on two platforms (64-bit AMD x86 and 32-bit ARM Cortex-M4 MC) for three bit-lengths. The duration and the clock cycles are approximated based on the number of iterations and the cost of individual operations. Two cases are considered: all signatures in the batch share the same public key and the general case of multiple keys.

| | | Time (AMD) | | Cycles (MC) | |
|---|---|---|---|---|---|
| | $t$ | Fixed key | Multiple keys | Fixed key | Multiple keys |
| GLV | 256 | 0.22ms | 0.38ms | 3 919 026 | 6 666 850 |
| | 384 | 2.03ms | 3.49ms | 39 809 138 | 67 946 960 |
| | 521 | 4.71ms | 8.10ms | 92 734 489 | 159 409 414 |
| GLV-3 | 256 | 0.23ms | 0.39ms | 3 841 371 | 6 697 835 |
| | 384 | 1.61ms | 3.10ms | 31 233 667 | 60 285 290 |
| | 521 | 3.62ms | 7.08ms | 71 272 985 | 139 403 489 |

considered batch of size 64 signatures. The average number of iterations turned out to be, on average, 3352, 5012, and 6788 for scalars of bit-length 256, 384, and 521, respectively.

The GLV method can be used to decompose each of the $B+2$ scalars, resulting in twice the amount $(2B+4)$ of scalars, but half the size. For a batch of size 64, the average number of iterations was then reduced to 2958, 4400, and 5925 for the individual bit-lengths in our simulation. This resulted in roughly 12% speed-up when compared to simple Bos-Coster for all three bit-lengths.

Our GLV-3 method can be adapted to batch signatures as well. For each equation $z_i u_i G + z_i v_i Q = z_i R_i$, we find $l_{i,0}, l_{i,1}, l_{i,2}$ such that $l_{i,0} + l_{i,1}\lambda_\varphi + l_{i,2} z_i = 0 \pmod n$ using Algorithm 4. Substituting into the verification equation, we get

$$l_{i,2} z_i u_i G + l_{i,2} z_i v_i Q + l_{i,0} R_i + l_{i,1} \varphi(R_i) = \infty,$$

with the following sum to be verified:

$$\left(\sum_{i=1}^{B} l_{i,2} z_i u_i\right) G + \left(\sum_{i=1}^{B} l_{i,2} z_i v_i\right) Q + \sum_{i=1}^{B} l_{i,0} R_i + \sum_{i=1}^{B} l_{i,1} \varphi(R_i) = 0.$$

The scalar corresponding to $G$ can be further split into three parts as in the previous sections assuming that we have two precomputed multiples of $G$. We can do the same trick for the point $Q$ and the corresponding scalar, but here we have to compute the multiples of $Q$ for each batch corresponding to $Q$, and so the cost of this must be included as well. The left side contains $2B + 6$ scalars of size similar to $n^{1/3}$.

For a batch of size 64, we found that the average number of additions (including the precomputation for $Q$) is then reduced to 2192, 3260, and 4376 for the individual bit lengths. We can use the measurements on both platforms to compare these batch versions of GLV-3 and GLV. If we multiply the number of iterations with the average measured cost of point addition, we get the approximate cost of GLV in Table 5 (columns 'Fixed key'). The cost of GLV-3 can be approximated in the same way with the addition of the cost of the lattice reduction measured in Table 2. The speed-up of GLV-3 for batch signatures is similar to the single-signature case. For 256 bits, the speed-up is negligible on the MC (2%) and even negative on AMD, but for 384 and 521 bits, it reaches $20 - 23\%$ on both platforms.

Now, we will remove the assumption of a fixed public key $Q$. Consider a public key $Q_i$ for every verification equation. The linear combination of the equations is then

$$\left(\sum_{i=1}^{B} z_i u_i\right) G + \sum_{i=1}^{B} z_i v_i Q_i - \sum_{i=1}^{B} z_i R_i = \infty.$$

Experiments showed that Bos-Coster takes approximately 5705, 8532, and 11556 iterations to compute the left side with $2B + 1$ scalars for individual bit-lengths. The GLV method doubles the number of scalars (with half the size), resulting in 5108, 7588, and 10229 iterations, i.e., $10 - 11\%$ speed-up.

Applying the GLV-3 method is not as straightforward as with the fixed public key. Using the same substitution with $l_{i,0} + l_{i,1}\lambda_\varphi + l_{i,2}z_i = 0 \pmod n$, we get

$$\left(\sum_{i=1}^{B} l_{i,2}z_iu_i\right) G + \sum_{i=1}^{B} l_{i,2}z_iv_iQ_i + \sum_{i=1}^{B} l_{i,0}R_i + \sum_{i=1}^{B} l_{i,1}\varphi(R_i) = \infty,$$

where $l_{i,0}$ and $l_{i,1}$ are small scalars ($\approx n^{1/3}$), but $l_{i,2}z_iv_i \approx n$. We can, however, decompose each $l_{i,2}z_iv_i$ using $\lambda_\varphi$ to two scalars of half the size ($\approx n^{1/2}$). As a result, we have $2B + 3$ scalars of similar size as $n^{1/3}$ (corresponding to $R_i, \varphi(R_i)$ and $G$) and $2B$ scalars of similar size as $n^{1/2}$ (corresponding to $Q_i$). Because of this gap between the two sets of the scalars, in one iteration of the Bos-Coster, the largest scalar $k_1 \approx n^{1/2}$ can be much bigger than the second largest $k_2 \approx n^{1/3}$. Hence, a simple scalar multiplication by $k_2$ might be needed to reduce $k_1$. Otherwise, the technique works the same as with scalars of the same size.

Table 5 (columns 'Multiple keys') summarizes the cost approximated in the same way as for the case with fixed $Q$. On both platforms, there is no speed-up for 256 bits and $11 - 13\%$ speed-up for the higher bit-lengths.

## 3.5 4-dimensional GLV

The GLS method described in Section 2.2 uses two endomorphisms for the 4-dimensional decomposition. Our GLV-3 method also uses two endomorphisms. This section describes why we did not focus on the 4-dimensional alternative.

For the VSM equality $kQ = R$ on a curve with endomorphism $\varphi$, we have considered $k$ as an endomorphism for which $kQ$ has been computed for us. Similarly, we can consider $k\varphi(Q) = \varphi(R)$ and find $l_0, l_1, l_2, l_3 \approx n^{1/4}$ satisfying:

$$\begin{aligned} l_0 + l_l\lambda_\varphi + l_2k + l_3k\lambda\varphi = 0 \pmod n \\ l_0Q + l_1\varphi(Q) + l_2R + l_3\varphi(R) = \infty \end{aligned}$$

with the assumption that $kQ = R$. In principle, this gives us a verification equation[4] with smaller scalars than GLV-3. Using two Straus-Shamir tricks with two pairs of scalars $l_0, l_1$ and $l_2, l_3$ in the JSF form, we can compute the left side with the cost $(\frac{t}{4}+2)A+\frac{t}{4}D+4M = (4.5t+26)M$. Compared to the GLV-3 method, this brings only $4-7\%$ speed-up. We can also precompute all possible combinations $j_0Q + j_1\varphi(Q) + j_2R + j_3\varphi(R)$, $j_i \in \{0, \pm 1\}$ and consider all four scalars in the JSF form with the join Hamming weight $0.64 \cdot \frac{t}{4} = \frac{4t}{25}$. The precomputation then costs $34A + 4M$ and the overall cost is

$$\left(34 + \frac{4t}{25}\right) A + \frac{t}{4}D + 4M \approx (3.51t + 378)M,$$

which is actually worse than GLV-3 for $t = 256$ and 10% better for $t = 521$.

The main issue of this 4-dimensional approach is the search for the short vector $(l_0, l_1, l_2, l_3)$ in a 4-dimensional lattice. Our experiments showed that Lagrange's algorithm adapted to 4 dimensions very often does not find a short vector of norm $\approx n^{1/4}$. Other algorithms could be used, e.g., [NS09], but we leave this as a future work. We could also apply this idea to VSS, resulting in eight scalars. Based on the observations with VSM, we decided that the overhead of the precomputation and the lattice reduction is too troublesome and most likely not worth the speed-up.

[4] It should be noted that if the second equation holds, then $kQ = R$ is true (and gets correctly verified) as long as $l_2 + l_3\lambda_\varphi \neq 0 \pmod n$, but this is expected.

# 4 The dependent coordinates problem

The GLV-3 method uses a fast endomorphism and a precomputed equality $kP = R$ to decompose the scalar $k$ into three smaller scalars. Smaller scalars imply faster scalar multiplication, which increases the presented speed. Beyond fast scalar multiplication, this decomposition also finds another application. This section completely changes the context and moves away from the verification of signatures into the field of side-channel attacks. We show that the GLV-3 technique can be used to significantly reduce the complexity of a specific type of side-channel attack.

Side-channel attacks are a common threat to ECC protocols implemented on embedded systems. We will consider certain types of side-channel attacks called zero-value point (ZVP) attacks [AT03]. ZVP attacks target a static private key during scalar multiplication where the attacker has control over the input point for the multiplication (e.g., ECDH). The attack works with the assumption that operations involving zero values are distinguishable from general random operations using power or EM analysis. The full context and the details can be found in the original paper [AT03].

We are interested in only one step of the attack — solving the dependent coordinates problem (DCP)[5] defined as follows:

*(DCP)* Given a scalar $k \in \mathbb{Z}$ and a polynomial $f \in \mathbb{F}_p[x_1, x_2]$, find two points $P, Q$, such that $f(x_P, x_Q) = 0$ and $Q = kP$.

Solving DCP efficiently is an open problem, but there are two cases, in which the problem can be efficiently solved. Firstly, for small scalars $k$ (at most 20 bits), one can generate the multiplication-by-$k$ map and substitute it into $f$, which results in a univariate polynomial equation in $x_P$ with a degree $\approx k^2 \deg f$ that is then solved for $x_P$. Secondly, the problem is also easy when $f$ is already a univariate polynomial (in either $x_P$ or $x_Q$). Nevertheless, for general $k$ and $f$, the best approach is to guess the point $P$ with the complexity $O(n)$. As a result, the ZVP attack is very limited by the difficulty of DCP, and only a few bits of the private key can be recovered.

Suchánek et al. [SSS25] realized that an implementation that uses the GLV method for scalar multiplication exposes more bits of the private. The DCP is, in that case, defined for $k = k_0 + k_1\lambda_\varphi$, where $k_0, k_1$ are small for the first several iterations. This allows the attacker to target both $k_0$ and $k_1$ at the same time, recovering almost twice as many bits as the original ZVP. While this result increases the number of recovered bits, the asymptotical complexity of the general DCP remains to be $O(n)$. The method uses multiplication-by-$k_i$ maps of degree $k_i^2$. Since for general $k$ we should expect $k_i^2 \approx n$, the complexity is at best $O(n)$.

We will now show how the 4-dimensional GLV method described in Section 3.5 can be used to reduce the asymptotical complexity of DCP on a GLV curve to $O(\sqrt{n}\log^3 n)$. Let $Q, P$ be the DCP solution points and $l_0, l_1, l_2, l_3 \approx n^{1/4}$ such that

$$l_0 + l_1\lambda_\varphi + l_2 k + l_3\lambda_\varphi k = 0 \pmod{n}.$$

. These can be found using, e.g., the LLL algorithm. The idea is to follow the symbolical approach described above, but instead of using a multiplication-by-$k$ map of degree $n^2$, we use the multiplication-by-$l_i$ maps of degree $\sqrt{n}$. Specifically, as in [SSS25], we use the Semaev polynomial [Sem04] that satisfies $S_4(x_{P_1}, x_{P_2}, x_{P_3}, x_{P_4}) = 0$ for any $P_1 \pm P_2 \pm P_3 \pm P_4 = \infty$. In our case with $l_0P + l_1\varphi(P) + l_2Q + l_3\varphi(Q) = \infty$, we get:

$$S(x_P, x_Q) = S_4(r_0(x_P), r_1(x_{\varphi(P)}), r_2(x_Q), r_3(x_{\varphi(Q)})) = 0,$$

where $r_0, r_1, r_2, r_3$ are the $x$-coordinates of the multiplication maps for $l_0, l_1, l_2, l_3$, respectively. The degree of $S$ is given by $\deg S_4 = 16$ and by the degrees of the $r_i$ maps, which

[5] The problem was formalized and named in [SCJ+21].

is $\approx \sqrt{n}$. Together with $f(x_P, x_Q) = 0$, we have two polynomial equations with two unknowns $x_P, x_Q$. It remains to compute the resultant, a univariate polynomial in $x_P$, and find its roots. Assuming fast integer arithmetic, this can be done in $O(\sqrt{n}\log^3 n)$ operations — a significant reduction from the original $O(n)$ complexity.

# 5 Conclusion

Our work shows that endomorphisms that have been used so far for 2-dimensional decomposition can also be used for higher dimensions in the right context. This improves the verification of Schnorr-like signature schemes. Specifically, experiments demonstrate a $\approx 20\%$ speed-up for single signature verification and 13% speed-up for batch verification. The improvement is mainly relevant for curves with higher bit-lengths with endomorphisms, including the classical pairing-friendly curves, such as BLS12-381 [Bow17]. In addition, the technique applies to any protocol involving the verification of scalar multiplication, such as the Blind Diffie-Hellman protocol, where we show 27% speed up.

The three dimensions in our GLV-3 technique came with the price of reducing 3-dimensional lattices. So as a byproduct of our work, we analyzed 3-dimensional lattice reduction algorithms and concluded that the simple adaption of Lagrange's algorithm for 3 dimensions is efficient enough and even faster than Semaev's algorithm with the state-of-the-art complexity.

Finally, to demonstrate that the idea of the GLV-3 extends beyond a signature verification, we lower the asymptotical complexity of the hard DCP problem that is hindering the so-called Zero-value point attack.

We leave two open directions for future work. First, this work considered Schnorr signatures, modified ECDSA signatures, and the BDHKE protocol. Other schemes, not necessarily signatures, could benefit from the GLV-3 method. There are only two requirements for the method: 1) the curve must admit an efficient endomorphism, and 2) scalar multiplication is verified in the scheme. As we discussed in Remark 1, the EdDSA scheme over a twisted Edwards curve with complete formulas could take advantage of the method as long as an appropriate curve with an efficient endomorphism was found.

Finally, we showed that while the GLV-3 method can be extended to four dimensions, the speed-up was at most 10% when compared to GLV-3. This does not include the cost of the 4-dimensional lattice reduction, and so the actual speed-up is lower. To tackle the 4-dimensional lattices, an analysis of the available reduction algorithm would be necessary. An appropriate choice might be the greedy algorithm from [NS09].

# References

[ABG+05] Adrian Antipa, Daniel Brown, Robert Gallant, Rob Lambert, Rene Struik and Scott Vanstone. Accelerated verification of ecdsa signatures. In *International Workshop on Selected Areas in Cryptography*, pages 307–318. Springer, 2005.

[AT03] Toru Akishita and Tsuyoshi Takagi. Zero-value point attacks on elliptic curve cryptosystem. In Colin Boyd and Wenbo Mao, editors, *Information Security*, pages 218–233, Berlin, Heidelberg. Springer Berlin Heidelberg, 2003. ISBN: 978-3-540-39981-0.

[Bab86] László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

[BBJ+08] Daniel J Bernstein, Peter Birkner, Marc Joye, Tanja Lange and Christiane Peters. Twisted edwards curves. In *Progress in Cryptology–AFRICACRYPT 2008: First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings 1*, pages 389–405. Springer, 2008.

[BDL+12] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012.

[BL07] Daniel J. Bernstein and Tanja Lange. Explicit-formulas database, 2007. URL: http://hyperelliptic.org/EFD.

[BLS01] Dan Boneh, Ben Lynn and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

[Bow17] Sean Bowe. Bls12-381: new zk-snark elliptic curve construction. https://electriccoin.co/blog/new-snark-curve/, 2017. (Visited on 16/01/2025).

[Bru15] Billy Bob Brumley. Faster software for fast endomorphisms. In *Constructive Side-Channel Analysis and Secure Design: 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers 6*, pages 127–140. Springer, 2015.

[BS64] Richard Bellman and EG Straus. Addition chains of vectors (problem 5125). *The American Mathematical Monthly*, 71(7):806–808, 1964.

[CFA+12] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2012. ISBN: 1439840008.

[De 94] Peter De Rooij. Efficient exponentiation using precomputation and vector addition chains. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 389–399. Springer, 1994.

[Fir24] Trusted Firmware. Mbed tls. https://github.com/Mbed-TLS/mbedtls, 2024.

[GLS09] Steven D. Galbraith, Xibin Lin and Michael Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 518–535, Berlin, Heidelberg. Springer Berlin Heidelberg, 2009. ISBN: 978-3-642-01001-9.

[GLV01] Robert P. Gallant, Robert J. Lambert and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 190–200, Berlin, Heidelberg. Springer Berlin Heidelberg, 2001. ISBN: 978-3-540-44647-7.

[HMV03] Darrel Hankerson, Alfred J. Menezes and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, Berlin, Heidelberg, 2003. ISBN: 038795273X.

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[Nak08] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system. *Satoshi Nakamoto*, 2008.

[NMV+95] David Naccache, David M'RaÏhi, Serge Vaudenay and Dan Raphaeli. Can dsa be improved?—complexity trade-offs with the digital signature standard—. In *Advances in Cryptology—EUROCRYPT'94: Workshop on the Theory and Application of Cryptographic Techniques Perugia, Italy, May 9–12, 1994 Proceedings 13*, pages 77–85. Springer, 1995.

[NS09] Phong Q Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. *ACM Transactions on algorithms (TALG)*, 5(4):1–48, 2009.

[Ope] OpenCash. Cashu. https://cashu.space/.

[Pro03] John Proos. Joint sparse forms and generating zero columns when combing. Technical report, Technical Report CORR 2003-23, Department of Combinatorics and Optimization, 2003.

[SCJ+21] Vladimir Sedlacek, Jesús-Javier Chi-Domínguez, Jan Jancar and Billy Bob Brumley. A formula for disaster: a unified approach to elliptic curve special-point-based attacks. In *Advances in Cryptology – ASIACRYPT 2021*. Springer, 2021. ISBN: 978-3-030-64837-4.

[Sem01] Igor Semaev. A 3-dimensional lattice reduction algorithm. In *Cryptography and Lattices: International Conference, CaLC 2001 Providence, RI, USA, March 29–30, 2001 Revised Papers*, pages 181–193. Springer, 2001.

[Sem04] Igor Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. Cryptology ePrint Archive, Paper 2004/031, 2004. URL: https://eprint.iacr.org/2004/031. https://eprint.iacr.org/2004/031.

[Sol01] Jerome A Solinas. Low-weight binary representation for pairs of integers. *Centre for Applied Cryptographic Research, University of Waterloo, Combinatorics and Optimization Research Report CORR 2001-41*, 2001.

[SSS25] Vojtěch Suchánek, Vladimír Sedláček and Marek Sýs. Decompose and conquer: ZVP attacks on GLV curves. Cryptology ePrint Archive, Paper 2025/076, 2025. URL: https://eprint.iacr.org/2025/076.

[Wag96] David Wagner. Chaumian ecash without rsa. https://cypherpunks.venona.com/date/1996/03/msg01848.html, 1996.

[Was08] Lawrence C Washington. *Elliptic curves: number theory and cryptography*. CRC press, 2008.

[WNR20] Pieter Wuille, Jonas Nick and Tim Ruffing. Schnorr signatures for secp256k1. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki, 2020.

[Wui13] Pieter Wuille. libsecp256k1. https://github.com/bitcoin-core/secp256k1, 2013.