Eccfrog512ck2: An Enhanced 512-bit Weierstrass Elliptic Curve

Víctor Duarte Melo and William J. Buchanan

Blockpass ID Lab, Edinburgh Napier University, Edinburgh

Abstract. Whilst many key exchange and digital signature methods use the NIST P256 (secp256r1) and secp256k1 curves, there is often a demand for increased security. With these curves, we have a 128-bit security. These security levels can be increased to 256-bit security with NIST P-521 Curve 448 and Brainpool-P512. This paper outlines a new curve - Eccfrog512ck2 - and which provides 256-bit security and enhanced performance over NIST P-521. Along with this, it has side-channel resistance and is designed to avoid weaknesses such as related to the MOV attack. It shows that Eccfrog512ck2 can have a 61.5% speed-up on scalar multiplication and a 33.3% speed-up on point generation over the NIST P-521 curve.

1 Introduction

Around 1985, Neal I. Koblitz [1] and Victor Miller [2] independently invented ECC. Their idea came from preliminary work conducted by H.W Lenstra Jr on using elliptic curves to crack the RSA method. His paper was entitled *Factoring Integers using Elliptic Curves* [3]. Although they created ideas between 1985 and 1987, the main methods of ECC did not take off until 2005. While NIST P-256 has become popular in TLS communications and secp256k1 is popular for blockchain applications, in some circumstances, there is a need for enhanced security, such as with NIST P-521 Curve 448 and Brainpool-P512. This paper adds the Eccfrog512ck2 curve, and which improves performance over the NIST P-521 curve.

2 Basics of Elliptic Curve Cryptography

Overall, there are three main curves used in production. With an Edwards curve, we have the form of:

$$x^2 + y^2 = 1 + d.x^2y^2 \pmod{p} \tag{1}$$

and is named after Harold Edwards. An example of this curve is Ed25519. For a Montgomery curve, we have the form:

$$By^2 = x^3 + Ax^2 + x \pmod{p} \tag{2}$$

and named after Peter Montgomery. With the Weierstrass curve, we have the form:

$$y^2 = x^3 + bx + c \pmod{p} \tag{3}$$

and which is used with Bitcoin. This elliptic curve is defined with p, a, b , g_x, g_y, and n, and where (g_x,g_y) is a base point on our curve, and n is the order of the curve. The curve used in Bitcoin and Ethereum is secp256k1, and which has the form of:

$$y^2 = x^3 + 7 \pmod{p} \tag{4}$$

and where $p = 2^{256} - 2^{32} - 977$. We can also use the NIST P256 (secp256r1) curve or the NIST-defined P-521 curve.

3 Curve design

EccFrog512CK2 is a custom-designed 512-bit elliptic curve and is aimed at providing robust cryptographic security.

3.1 Prime field

The curve uses a 512-bit prime field and has a precisely selected prime modulus, ensuring heightened resistance to classical and potential quantum cryptographic threats.

3.2 Deterministically generated coefficient

The curve has a deterministically generated coefficient (b) and which is computed using the cryptographically secure BLAKE3 hash function. This ensures reproducibility and verifiable integrity.

3.3 Curve equation

The curve equation used is:

$$y^2 \equiv x^3 - 7x + b \pmod{p} \tag{5}$$

and where p is the carefully chosen 512-bit prime modulus. In terms of security validations and characteristics. The following sections define these.

3.4 Non-singularity

Verified via discriminant computation, ensuring:

$$\Delta \neq 0 \tag{6}$$

3.5 Prime Order

Fully verified prime order curve, eliminating small subgroup vulnerabilities.

3.6 Resistance to MOV and Twist Attacks

The curve has extensive checks confirming immunity to common elliptic curve vulnerabilities. This includes the MOV and Twist Attacks [4]. The MOV attack was outlined by Menezes and Vanstone [5] and defined how we can reduce the strength of the elliptic curve method to a discrete logarithm problem. It uses key pairing where we have two cyclic groups (G_1 and G_2), and which are of an order of a prime number (n). A pairing on (G_1,G_2,G_T) defines the function $\hat{e} : G_1 \times G_2 \rightarrow G_T$, and where g_1 is the generator for G_1 and g_2 is the generator for G_2. If we have the points of P on G_1 and Q on G_2, we get a pairing of:

$$e^(P, Q) \tag{7}$$

Now, if we select a private key value of x, the public key will become:

$$P_{pub} = x \cdot P \tag{8}$$

In order to find x, we would have to search the values of x to match P to x. In pairing, we can reduce the difficulty with:

$$\hat{e}(xP, Q) = \hat{e}(P, Q)x \tag{9}$$

This now becomes a discrete logarithm problem within a finite field, and which makes it easier to find x [6].

3.7 Scalar Multiplication Techniques

The curve has been implemented using wNAF (windowed Non-Adjacent Form), the Montgomery Ladder, and GLV methods for performance optimization and side-channel attack resistance [7].

3.8 Side-Channel Resistance

The curve uses constant-time operations and secure memory handling in order to prevent timing and cache-based attacks [7].

4 Methodology

The core parameters defined for the curve are defined in Table 1 and Table 2 outlines the parameters selected. The code for validation is then given in Listing 1.1. Table 3 outlines the general security properties of the curve, and Table 4 outlines the comparison with P-521 (secp521r1) and Curve 25519. The processor used for the tests is AMD Ryzen 9 5950X with 3.4 GHz base clock. The code for the curve is defined at [8].

Table 1: Overview of the curve parameters and structural choices

Property	**EccFrog512CK2**
Prime field size	512 bits
Field type	Prime field $\mathbb{F}_p$
Curve form	Short Weierstrass
Coefficient a	$-7 \mod p$
Coefficient b	Deterministic via BLAKE3
Curve order	Verified prime
Cofactor	1
Discriminant Δ	$\neq 0$ (non-singular)
Twist security	Verified
MOV security	Resistant

Table 2: Parameters for curve

Parameter	Value
p	9149012705592502490164965176888130701548 0539186997936896723448077728011058306814 9878074662253072941885847710307359191805 8480028776841126664954537807339721
a	p - 7
b	9586418985095791770393300613179378564924 0252916618759767550461391845895018181
n	9149012705592502490164965176888130701548 0539186997936896723448077728011058305572 6912325585091574506354113315750370728404 8429261692283957712127567713136519
G_x	8426241697659200371183582771153260966569955 6 9961504423264097242343194706012957373611229 8 7449773324161750213370827758560580583947862 6 4506901662703740544432
G_y	4970129934163735248083452609809843496231929 6 2041903848950639136613618648599428832075866 8 1727900608018098106881920821464319706831135 5 723943357001111255600 1

Table 3: Cryptographic security properties of the curve.

Security Aspect	**EccFrog512CK2**
Subgroup attacks	Not possible (cofactor = 1)
Curve validation	Transparent generation
Side-channel resistance	Constant-time ops
Field structure	No special form
Hash-to-curve support	Planned

Table 4: High-level comparison between EccFrog512CK2 and well-known ECC curves

Feature	**EccFrog512CK2**	**P-521**	**Curve25519**
Bits of security	∼256	∼256	∼128
Transparency	Yes	No	Yes
Prime field	512-bit	521-bit	255-bit
Implementation freedom	Full control	NIST only	Medium
Open-source origin	Community	Government	Yes

5 Evalution

The evaluation of the timing of the operations for point generation, scalar multiplication and ECDH key exchange is defined in Table 5. We can see a significant speed for all the areas, including a 61.5% speed-up on scalar multiplication and a 33.3% speed-up on point generation. Table 6outlines the timing for the important ECC operations, such as key pair generation, shared secret generation, signature generation and signature verification. The evaluation of the Eccfrog512ck2 curve against NIST P-521 is defined in Figure 1.

Table 5: Time taken for core ECC operations

Operation	EccFrog512CK2 (ms)	NIST P-521 (ms)	Speed-up (%)
Point Generation	0.4	0.6	33.3
Scalar Multiplication	0.826	2.143	61.5
Point Validation	0.097	0.15	35.3
ECDH Key Exchange (total)	1.81	3.2	43.4

6 Conclusions

Eccfrog512ck2 provides a significant enhancement in performance and security over NIST P-521 and could be used in applications which require enhanced security levels over NIST P-256 and secp256k1.

7 Appendix

Table 6: Approximate timing benchmarks for key ECC operations

Metric	**EccFrog512CK2**
Keypair generation (ms)	∼0.85 ms
Shared secret derivation (ECDH)	∼1.81 ms
Signature generation (est.)	∼0.95 ms
Signature verification (est.)	∼1.20 ms

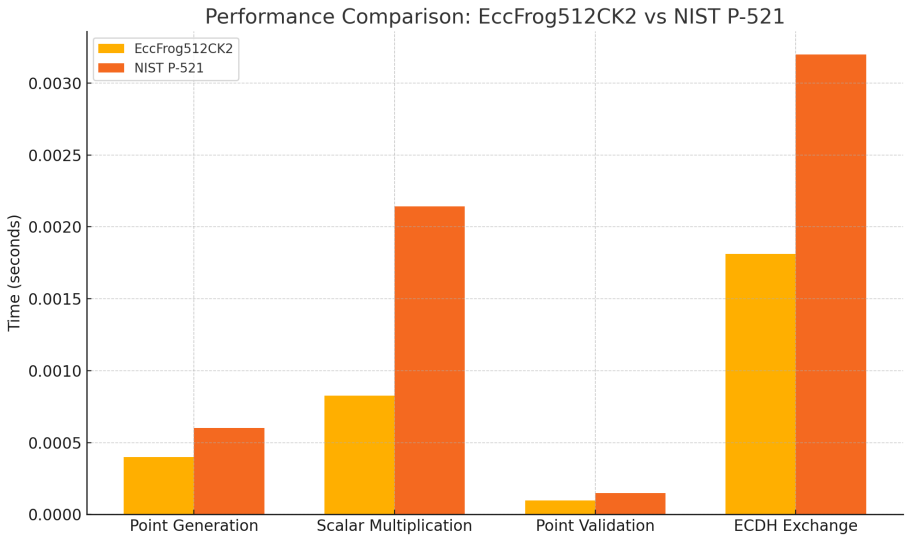

Fig. 1. Performance comparison between NIST P-521

Listing 1.1. SageMath script to verify EccFrog512CK2

```
# parameters
p = Integer("914901270559250249016496517688813070154805391869979368967234480777280110583068149878074662253072941885847710307359191805848002877684112666495453780733972")
a = p - 7
b = Integer("958641898509579177039330061317937856492402529166187597675504613918458950181181")
n = Integer("914901270559250249016496517688813070154805391869979368967234480777280110583055726912325585091574506354113315750370728404842926169228395771212756771313651 9")

F = FiniteField(p)
E = EllipticCurve(F, [a, b])

discriminant = E.discriminant()
hasse_bound = 2 * sqrt(p)
within_hasse = abs(n - (p + 1)) <= hasse_bound
non_singular = discriminant != 0

print("=== EccFrog512CK2 Cryptographic Validation ===")
print("Prime p (bits):", p.nbits())
print("Coefficient a:", a)
print("Coefficient b:", b)
print("Curve order n is prime:", n.is_prime())
print("Discriminant is non-zero:", non_singular)
print("Hasse's theorem bound satisfied:", within_hasse)
print("2 sqrt(p) (Hasse bound):", hasse_bound)
print("|n - (p + 1)|:", abs(n - (p + 1)))
```

References

1. N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.
2. V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques*. Springer, 1985, pp. 417–426.
3. H. W. Lenstra Jr, "Factoring integers with elliptic curves," *Annals of mathematics*, pp. 649–673, 1987.
4. djb, "SafeCurves: Twist security — safecurves.cr.yp.to," https://safecurves.cr.yp.to/twist.html, [Accessed 11-04-2025].
5. A. Menezes, S. Vanstone, and T. Okamoto, "Reducing elliptic curve logarithms to logarithms in a finite field," in *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, 1991, pp. 80–89.

6. W. J. Buchanan, “Cracking elliptic curves with the mov attack using kryptology,” https://asecuritysite.com/bls/mov, Asecuritysite.com, 2025, accessed: April 10, 2025. [Online]. Available: https://asecuritysite.com/bls/mov
7. V. Meloasm and W. Buchanan, “GitHub - victormeloasm/OpenFrogget: OpenFrogget is a secure file encryption tool combining custom elliptic curve eccfrog512ck2 cryptography with AES-GCM for hybrid encryption. Built in modern C++ with a fully open GPL license. — github.com,” https://github.com/victormeloasm/OpenFrogget, [Accessed 11-04-2025].
8. ——, “OpenFrogget/src/eccfrog512ck2.cpp at main · victormeloasm/OpenFrogget — github.com,” https://github.com/victormeloasm/OpenFrogget/blob/main/src/eccfrog512ck2.cpp, [Accessed 13-04-2025].