The arithmetic of pairing-based proof systems

Youssef El Housni

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAX104

Thèse de doctorat

The Arithmetic of Pairing-Based Proof Systems

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Mathématiques et Informatique

Thèse présentée et soutenue à Palaiseau, le 18 novembre 2022, par

YOUSSEF EL HOUSNI

Composition du Jury :

Sylvain Duquesne Professeur, Université de Rennes 1 (IRMAR)	Président
Craig Costello Principal Researcher, Microsoft Research Redmond	Rapporteur
Sarah Meiklejohn Professor, University College London	Rapporteure
Laurent Imbert Directeur de recherche, CNRS (LIRMM)	Examinateur
Mary Maller Researcher, Ethereum Foundation	Examinatrice
Yannick Seurin Researcher, Ledger	Examinateur
François Morain Professeur, École Polytechnique (LIX)	Directeur de thèse
Aurore Guillevic Chargée de recherche, Inria (LORIA)	Co-directrice de thèse
Daniel Augot Directeur de recherche, Inria (GRACE)	Invité
Nicolas Liochon Head of R&D, ConsenSys	Invité

This work has been partially achieved within ConsenSys R&D. It has been partially funded by ConsenSys while I was a research engineer in the `gnark` team.

Acknowledgement

Je dois l'existence de cette thèse, et tout le succès que j'ai eu en tant que chercheur, à mes merveilleux directeurs de thèse: Daniel Augot, François Morain et Aurore Guillevic. Oui, j'en ai eu trois et j'en suis fier et reconnaissant. Ils ont investi énormément de foi et d'efforts en moi, et m'ont guidé à chaque étape de mon voyage de thèse. D'abord je tiens à remercier Daniel d'avoir cru en moi dès le premier jour et de m'avoir accompagner tout au long de ces trois dernières années. Je tiens à remercier François sans qui cette thèse n'aurait pu ni commencer ni finir. Ses explications au tableau de la CM, des courbes et isogénies étaient indispensables pour avancer. Enfin, je tiens à remercier Aurore pour son aide précieuse et son dévouement indéfectible. La meilleure chose qui m'est arrivée pendant ma thèse, c'est sans doute d'avoir croiser son chemin.

I would also like to thank my thesis committee members: Sylvain Duquesne, Craig Costello, Sarah Meiklejohn, Laurent Imbert, Mary Maller, Yannick Seurin and Nicholas Liochon. I am very fortunate and honored to have such amazing researchers in my committee. I really enjoyed all the fruitful discussions we had and hope these continue in the future.

When I started this thesis, I was working at EY in the Paris office. I would always remember that it was thanks to Quentin Drouot and Xavier De Boissieu that I had a chance to start this PhD program. They allowed me to enroll in this program while still working in parallel at EY. The firm didn't have a research culture and I was probably the only employee worldwide conducting a PhD research in parallel. This was exclusively thanks to the support of Quentin and Xavier. I would also like to thank my colleagues at EY: Sami, Ali, Pauline, Victorien, Lucas, Alexandre, Anne-Fleur, Beri, Eliot, Elliot... and the Ibiza team ;)

During my two last years, I was hired by ConsenSys where I'm still working as of today. ConsenSys not only accepted to hire me while I'm doing a PhD in parallel but also accepted to fund the PhD program. I would like to thank the people who made this possible: Julien, Nicholas and Gautam. I would also like to thank my teammates: Gautam Botrel, Thomas Piellard, Ivo Kubjas and Arya Tabaie. Thank you guys for your support, kindness and great spirit on a daily basis. We met only twice (where we had a lot of fun) but your remote support was always ubiquitous. My thanks also go for the Paris team: Alexandre, Olivier, Franklin, Azam...

Enfin, mais non des moindres, je tiens à remercier ma famille de m'avoir toujours soutenu. Je remercie particulièrement mon frère et ma sœur pour leurs encouragements constants. Enfin, du fond de mon cœur, je tiens à remercier mes parents pour leur amour inconditionnel et leur soutien indéfectible. Je leurs dois plus que je ne pourrais jamais exprimer avec des mots. C'est à eux que je dédie cette thèse.

بِسمِ اللّٰهِ الرَّحمٰنِ الرَّحِيمِ

«وَ اخفِض لَهُمَا جَنَاحَ الذُّلِ مِنَ الرَّحمَة وَ قُل رَبِّ ارحَمهُمَا كَمَا رَبَّيَانِي صَغيرًا»

لِوَالِدَيَّ احمَد وَ سُعَاد .

Résumé en Français

La grande majorité des services numériques proposés aujourd'hui sont fournis par des systèmes centralisés, qui concentrent l'autorité et les capacités entre les mains de quelques privilégiés. Si cette centralisation réduit la complexité de mise en œuvre et augmente l'efficacité, elle s'accompagne d'une résistance à la censure et d'un manque de transparence. Au cours des dernières années, les systèmes décentralisés, tels que les blockchains et les registres distribués, ont suscité un intérêt croissant. En simples termes, un registre distribué peut être considéré comme une méthode qui préserve les données dans une base de données distribuée tout en garantissant que toutes les parties honnêtes ont la même vue des données, même en présence de parties corrompues. La définition la plus élémentaire d'une blockchain est un registre numérique distribué qui enregistre de manière vérifiable les transactions entre plusieurs parties. En permettant aux utilisateurs de conserver une copie du registre et en synchronisant toutes les copies à l'aide d'un mécanisme de consensus, elle élimine le besoin de vérification par une autorité centrale.

Toutefois, ces avantages se font souvent au détriment de deux propriétés essentielles : la confidentialité et la mise à l'échelle. Pour garantir l'exactitude des calculs, les systèmes de registres décentralisés existants exigent que les parties publient l'intégralité de leur état de calcul, qui est ensuite vérifié par les autres parties qui doivent ré-exécuter le calcul. Du point de vue de la confidentialité, cela révèle le calcul, les données d'entrée et l'identité des parties. Du point de vue de la mise à l'échelle, cela signifie que le coût des calculs coûteux est supporté par chaque partie du système, et non par la seule partie qui invoque le calcul. Il convient de noter que, outre les blockchains, ces préoccupations sont pertinentes pour tout système qui doit fournir une preuve de calcul correct tout en préservant la vie privée, par exemple une entreprise fournissant des données à un cabinet d'audit.

Pour résoudre ce dilemme, les systèmes de preuve à divulgation nulle de connaissance se sont révélés être une solution clé. Un système de preuve est un protocole interactif dans lequel une partie (appelée le prouveur) essaie de convaincre une autre partie (appelée le vérifieur) qu'un énoncé donné est vrai. Dans une preuve à divulgation nulle de connaissance, nous exigeons en outre que la preuve ne révèle rien d'autre que la vérité de l'énoncé. Une preuve est non-interactive si aucune communication n'est requise entre le prouveur et le vérifieur, sauf pour l'envoi de la preuve. Dans la classe des preuves non interactives, un concept particulièrement intéressant pour prouver l'intégrité de calcul est le "Succinct Non-

interactive ARgument of Knowledge" (SNARK). Il fournit une preuve calculatoirement consistante, peu coûteuse à vérifier et petite de taille par rapport à la taille de l'énoncé ou du témoin. Au lieu de publier un calcul coûteux sur la blockchain, une partie publie une preuve SNARK à divulgation nulle de connaissance de l'exécution correcte de ce calcul. Cela résout à la fois le problème de la confidentialité et celui de la mise à l'échelle.

Les SNARKs basés sur le couplage bilinéaire sont les systèmes de preuve les plus efficaces en ce qui concerne la propriété de succinctivité. En outre, cette propriété en fait de bons candidats pour la composition de preuves récursives. Une preuve récursive est une preuve faite par un prouveur qui convainc un vérifieur que d'autres preuves faites par d'autres prouveurs ont été correctement vérifiées par le prouveur. Cela permettrait à une seule preuve d'attester inductivement de l'exactitude de nombreuses preuves antérieures, ce qui aiderait les blockchains encore plus pour la mise à l'échelle.

Dans cette thèse, nous étudions l'arithmétique des systèmes de preuves récursives basés sur le couplage bilinéaire. Nous présentons une étude à trois étapes du processus : les courbes pour instancier un SNARK, les courbes pour instancier un SNARK récursif, et aussi les courbes pour exprimer un énoncé lié à une courbe elliptique. Nous fournissons de nouvelles constructions de courbes pour les SNARKs et de nouvelles familles de courbes à 2 chaînes pour les SNARKs récursifs. Nous dérivons et implémentons en open-source des algorithmes efficaces pour accélérer l'arithmétique sur ces courbes : effacement de cofacteur, test d'appartenance à un sous-groupe, multiplication multi-scalaire et couplages bilinéaires sur les 2-chaînes. Nous étudions et optimisons également l'arithmétique des courbes elliptiques et les couplages bilinéaires en tant qu'énoncé SNARK, ce qui permet la génération de preuves récursives la plus rapide.

Contributions

Conference proceedings

Optimized and secure pairing-friendly elliptic curve suitable for one-layer proof composition

Youssef El Housni and Aurore Guillevic

In S. Krenn, H. Shulman, and S. Vaudenay (Eds.): **CANS 2020**, LNCS 12579, pp. 259–279, 2020.
DOI: `https://doi.org/10.1007/978-3-030-65411-5_13`
Full verion: `https://eprint.iacr.org/2020/351`

Families of SNARK-friendly 2-chains of elliptic curves

Youssef El Housni and Aurore Guillevic

In O. Dunkelman and S. Dziembowski (Eds.): **EUROCRYPT 2022**, Part II, LNCS 13276, pp. 367–396, 2022.
DOI: `https://doi.org/10.1007/978-3-031-07085-3_13`
Full verion: `https://eprint.iacr.org/2021/1359`

Co-factor clearing and subgroup membership testing on pairing-friendly curves

Youssef El Housni, Aurore Guillevic and Thomas Piellard

In L. Batina and J. Daemen (Eds.): **AFRICACRYPT 2022**, LNCS 13503, pp. 518–536, 2022.
DOI: `https://doi.org/10.1007/978-3-031-17433-9_22`
Full verion: `https://eprint.iacr.org/2022/352`

Pairings in rank-1 constraint systems

Youssef El Housni

To appear in the 21st International Conference on Applied Cryptography and Network Security — **ACNS 2023**.
Full verion: `https://eprint.iacr.org/2022/1162`

Journals

A survey of elliptic curves for proof systems

Diego F. Aranha, *Youssef El Housni* and Aurore Guillevic

To appear in Designs, Codes and Cryptography journal – **DCC 2022** (special issue: the mathematics of zero-knowledge).
DOI: `https://doi.org/10.1007/s10623-022-01135-y`
Full verion: `https://eprint.iacr.org/2022/586`

Pre-prints

EdMSM: Multi-Scalar-Multiplication for recursive SNARKs and more

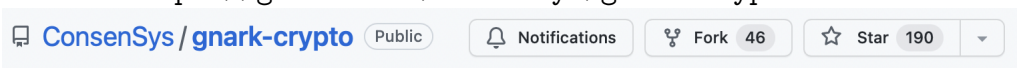
Youssef El Housni and Gautam Botrel

In submission.
Full verion: `https://eprint.iacr.org/2022/1400`

Software

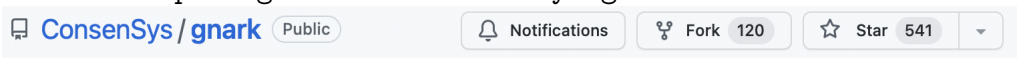
`gnark-crypto`: A fast Go library for cryptography

Gautam Botrel, Thomas Piellard, *Youssef El Housni*, Arya Tabaie and Ivo Kubjas
GitHub: `https://github.com/ConsenSys/gnark-crypto`



`gnark`: A fast Go library for zk-SNARKs

Gautam Botrel, Thomas Piellard, *Youssef El Housni*, Arya Tabaie and Ivo Kubjas
GitHub: `https://github.com/ConsenSys/gnark`



Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

BLS	Occurs as Barreto–Lynn–Scott (curve) and Boneh–Lynn–Shacham (signature)
BN	Barreto–Naehrig (curve)
BW	Brezing–Weng (curve)
CM	Complex Multiplication
CP	Cocks–Pinch (curve)
DL	Discrete Logarithm (problem)
ECDH	Elliptic Curve Diffie–Hellman (protocol)
EdDSA	Edwards Digital Signature Algorithm
FFT	Fast Fourier Transform
GLV	Galant–Lambert–Vanstone (method)
GMV	Galbraith–McKee–Valença (curve)
KSS	Kachisa–Schaefer–Scott (curve)
LLL	Lenstra–Lenstra–Lovász (algorithm)
MNT	Miyaji–Nakabayashi–Takano (curve)
MOV	Menezes–Okamoto–Vanstone (attack)
MSM	Multi-Scalar-Multiplication
NFS	Number Field Sieve (algorithm)
PLONK	Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge
R1CS	Rank-1 Constraint System
SNARK	Succinct Non-interactive ARgument of Knowledge
SSWU	Simplified Shallue-van de Woestijne-Ulas (method)
SW	Short Weierstrass (form)
tEd	twisted Edwards (form)
q-PKE	q-Power Knowledge of Exponent (assumption)
q-CPDH	q-Computational Power Diffie-Hellman (assumption)
ZK	Zero-Knowledge

Introduction

Motivation and outline

The vast majority of digital services offered today are delivered through centralized systems, which concentrate authority and ability in the hands of a select few. While this centralization reduces implementation complexity and increases effectiveness, it comes with censorship resistance and lack of transparency. Over the past several years there have been an increasing interest in decentralized systems, such as blockchains and distributed ledgers. In the simplest terms, a distributed ledger can be thought of as a method that preserves data across a distributed database while guaranteeing that all honest parties have the same view of the data, even in the presence of corrupt parties. The most basic definition of a blockchain is a digitally distributed ledger that verifiably records transactions between many parties. By enabling users to keep a copy of the ledger and syncing all copies using a consensus mechanism, it eliminates the need for verification by a central authority.

However, these benefits often come at the expense of two key properties: privacy and scalability. To ensure the correctness of computations, existing decentralized ledger systems require that parties publish their entire computational state, which is then checked by the other parties who have to re-execute the computation. From the privacy perspective, this reveals the computation, the input data and the identities of the parties. From the scalability perspective, this means that the cost of expensive computations is carried by every party in the system, as opposed to just the party invoking the computation. Note that, besides blockchains, these concerns are relevant to any system which has to provide a proof of correct computation while preserving privacy, for instance a company providing data to an audit firm.

To address this dilemma zero-knowledge proof systems have shown to be a key solution. A proof system is an interactive protocol where one party (called the prover) tries to convince another party (called the verifier) that a given statement is true. In a zero-knowledge proof, we further require that the proof does not reveal anything beyond the truth of the statement. A proof is non-interactive if no communication is required between the prover and the verifier except from sending the proof. In the class of non-interactive proofs, a particularly interesting concept for proving the computational integrity is the Succinct Non-interactive ARgument of Knowledge (SNARK). It provides a computationally sound proof, cheap to verify and small compared to the size of the statement or the witness. Instead of publishing an expensive computation on the blockchain, a party publishes a zero-knowledge SNARK proof of the correct execution of that computation. This solves both the privacy issue and the scalability issue.

Pairing-based SNARKs are the most efficient proof systems with respect to the succinctness property. Furthermore, this property makes them good candidates for recursive proof composition. A recursive proof is a proof made by a prover that convinces a verifier that other proofs made by other provers have been correctly verified by the prover. This would allow a single proof to inductively attest to the correctness of many former proofs, yielding even more scalable blockchains.

In this dissertation, we investigate the arithmetic of recursive pairing-based proof systems. We present a study at three stages of the process: curves to instantiate a SNARK, curves to instantiate a recursive SNARK, and also curves to express an elliptic-curve related statement. We provide new constructions of curves for SNARKs and new families of 2-chain curves for recursive SNARKs. We derive and implement in open-source efficient algorithms to speed up the arithmetic on these curves: co-factor clearing, subgroup membership testing, multi-scalar multiplication and pairings over 2-chains. We also study and optimize elliptic-curve arithmetic and pairings as a SNARK statement, yielding to the fastest recursive proof generation in pairing-based settings.

Pairing-friendly curves for SNARKs

Pairing-based SNARKs cannot be instantiated with generic-purpose elliptic curves, but instead require tailored constructions of elliptic curves. More precisely, they need pairing-friendly elliptic curves with additional properties, purposely designed to provide an efficient implementation. The proof generation involves solving multiple large instances of tasks about polynomial arithmetic in $\mathbb{F}_r[X]$ (where r is the curve prime subgroup order) and multi-scalar multiplication (MSM) over the pairing groups. The proof verification mainly involves computing a product of pairings.

In Part I (Chapters 1 and 2), we give preliminaries on pairings, pairing-friendly constructions and pairing-based proof systems. In Part II (Chapter 3), we first give an overview of the elliptic curves designed for different proof systems, revisit some constructions in terms of efficiency and security and propose some new ones. Next, we focus on efficient arithmetic over these curves. We derive new results on co-factor clearing and subgroup membership in the pairing groups.

Pairing-friendly curves for recursive SNARKs

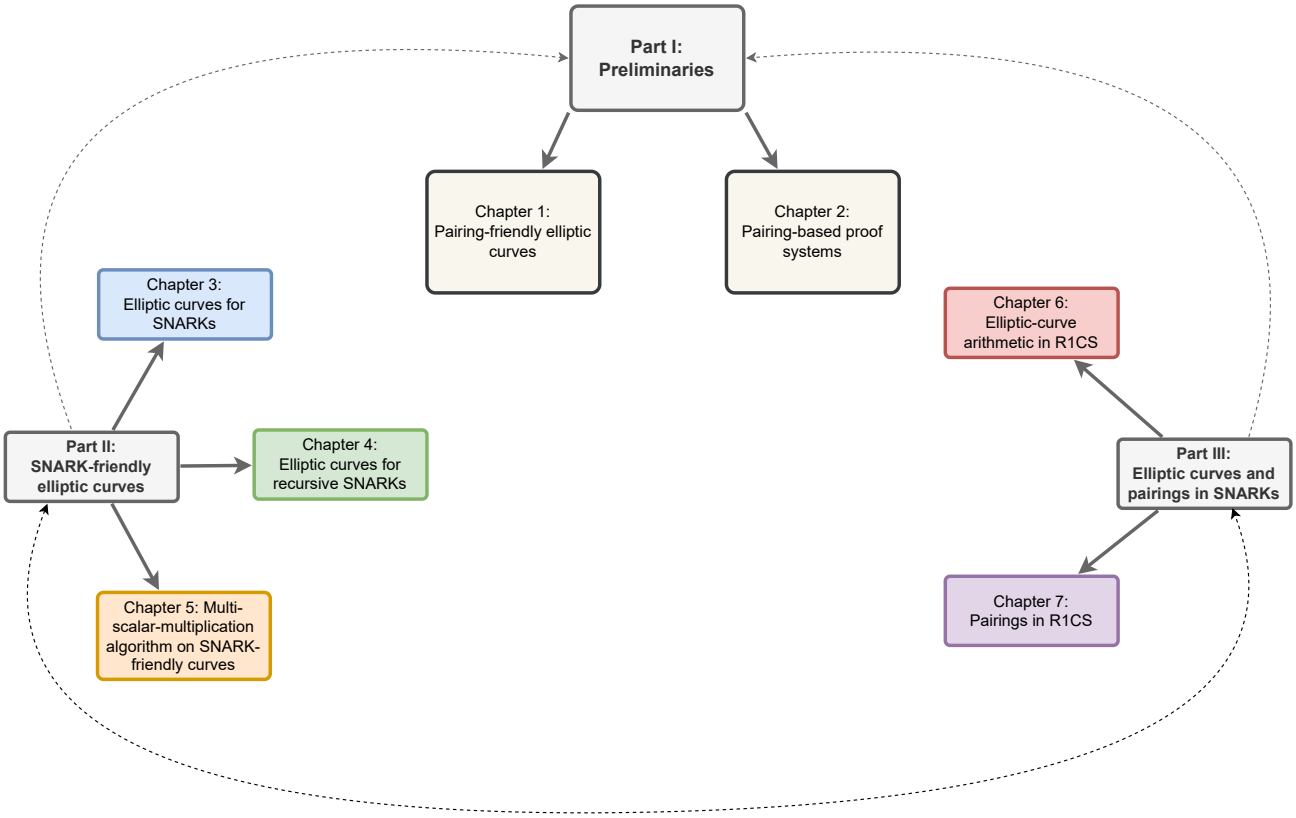
Because SNARKs are succinct they allow efficient proof composition. The goal of such proofs is to verify the validity of other proofs. This would allow a single proof to inductively attest to the correctness of many former proofs. However, once a first proof is generated, it is highly impractical to use the same elliptic curve to generate a second proof verifying the first one. A practical approach requires two different curves that are closely tied together.

In Part II (Chapter 4) we focus on the construction of families of cycles and chains of SNARK-friendly elliptic curves for recursive proof systems. First, we present results from the literature on cycles of pairing-friendly, plain and hybrid curves before presenting our results on families of 2-chains and their arithmetic. We focus on efficient arithmetic on the 2-chains and derive novel pairing algorithms. Next, in Part II (Chapter 5), we revisit the MSM algorithm and propose optimizations in the case of 2-chains.

Pairing-friendly curves inside SNARKs

While SNARKs allow proving general-purpose computations, in many applications these computations revolve around proving some cryptographic operations such as hashings, encryptions, key exchanges or signatures. These operations often require efficiently expressing elliptic-curve arithmetic as a SNARK computation which actually is a verification. This changes the perspective on the optimization of operations on curves. The elliptic curve used for this can be independent of the SNARK elliptic curve to prove the computation. We call it an associated curve.

In Part III (Chapter 6), we investigate the question of what associated elliptic curve is suitable for this problem in the light of the Rank-1 Constrain System (R1CS), a widely used model to express SNARK computations. We optimize the scalar multiplication algorithm in R1CS and construct suitable curves associated to the SNARK-friendly curves introduced in the previous chapters. Next, in Part III (Chapter 7), we consider efficiently implementing pairings in R1CS. This is a key step to speed up recursive proofs generation. We show that our techniques almost halve the arithmetic circuit depth of the previously best known pairing implementation on a BLS12 curve, resulting in 70% faster proving time. We also investigate the case of BLS24 curves.



Industrial Impact

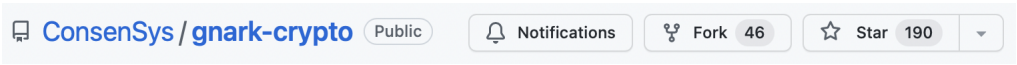
The work in this dissertation has resulted in an industrial impact. Below we first describe the `gnark` open-source ecosystem that implements the totality of this work, and then we elaborate on the adoption by other ecosystems of this work.

`gnark` ecosystem

The `gnark` ecosystem is composed of two libraries: `gnark-crypto` and `gnark`.

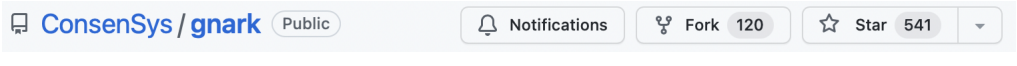
`gnark-crypto`. This is an open-source software library in Go that provides elliptic-curve and pairing-based cryptography on a wide variety of curves (BN, BLS, BW, Edwards). It also provides optimized finite field arithmetic and various algorithms of particular interest to zero- knowledge proof systems (FFT, MSM, KZG, MiMC, FRI). The curves and algorithms described in this dissertation (Chapters 3, 4 and 5) are all implemented in this library. The `gnark-crypto` code is maintained by ConsenSys under the Apache 2.0 license.

Source code `https://github.com/ConsenSys/gnark-crypto`



`gnark`. This is an open-source software library in Go that offers a high-level API to design SNARK circuits. It implements Groth16 and PLONK proof systems using any elliptic curve implemented in `gnark-crypto`. The PLONK implementation comes in two flavours: a universal one using KZG polynomial commitment and a transparent one using FRI polynomial commitment. The repository also comes with a standard library that implements SNARK-friendly circuits such as pairing-based proof composition, algebraic hashes (MiMC) and EdDSA signatures (on associated curves). The curves and algorithms described in this dissertation (Chapters 3, 4, 6 and 7) are all implemented in this library. The `gnark` code is maintained by ConsenSys under the Apache 2.0 license.

Source code `https://github.com/ConsenSys/gnark`



Documentation `https://docs.gnark.consensys.net`

Playground `https://play.gnark.io`

At the time of writing, the `gnark` ecosystem is used by several projects such as: ConsenSys zk-rollup, Algorand, Binance, Coinbase, IBM, Baseline, Geth, iden3 and Provide.

Other ecosystems

We also implemented some of the curves, algorithms and optimizations in this dissertation in other ecosystems: `libsnark` and `arkworks`.

`libsnark`. This is an open-source software SNARK ecosystem in C++ developed by SCIPR Lab. It uses underneath `libff` which is a C++ library for finite fields and elliptic curves. During the first year of this thesis, we contributed to the implementation of several elliptic curves and pairing algorithms (Chapters 3 and 4) in this fork[1] of `libff`. It was used by Ernst&Young in the Nightfall[2] product.

`arkworks`. This is an open-source software SNARK ecosystem in Rust (`http://arkworks.rs`). We contributed to the implementation of 2-chains and several optimizations (Chapters 3 and 4). In particular, our implementation of the BW6-761 curve in `arkworks` is now used by three blockchain projects: Celo (`https://celo.org`), Aleo (`https://aleo.org`) and Espresso Systems (`https://espressosys.com`).

[1] `https://github.com/EYBlockchain/zk-swap-libff`
[2] `https://github.com/EYBlockchain/nightfall`

I

Preliminaries

Chapter

1

Pairing-friendly elliptic curves

1.1 Background on elliptic curves

Let $\mathbb{F}_p$ be a field for some prime $p > 3$. In this thesis we shall always define curves over a field of prime order and of characteristic strictly greater than three. An elliptic curve E over such a field $\mathbb{F}_p$ can always be reduced to the short Weierstrass equation of the form

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

where $a, b \in \mathbb{F}_p$. Let $\Delta = 4a^3 + 27b^2$, the discriminant of the cubic equation in x. Then E is singular if $\Delta = 0$ (repeated roots) and nonsingular otherwise (distinct roots). Elliptic curves are nonsingular (smooth) curves.

For any field $\mathbb{F}_p$ define $E(\mathbb{F}_p)$ to be the set of all solutions of E over $\mathbb{F}_p$ called the finite points along with a special point denoted $\mathcal{O}$, that is called the point at infinity. We write $\#E(\mathbb{F}_p)$ for the number of elements of $E(\mathbb{F}_p)$. Solving the curve equation using projective coordinates, one can show that $\mathcal{O} = (0 : 1 : 0)$ is always a unique infinite solution to the equation. This set of points forms a group under the composition law noted additively $(+)$.

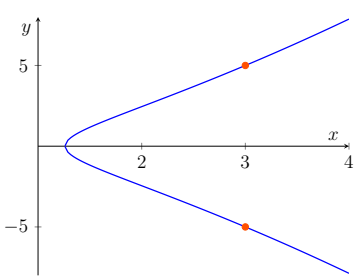
The group composition law: Chord-and-tangent

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be any two points on $E(\mathbb{F}_p)$. The group law is given as follows:

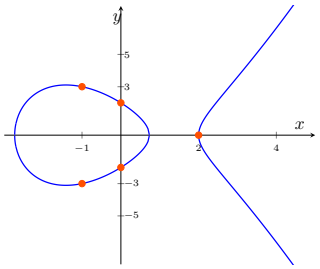
(a) $P_1 + \mathcal{O} = P_1$ and $\mathcal{O} + P_2 = P_2$

(b) $-P_1 = (x_1, -y_1)$ and $-P_2 = (x_2, -y_2)$

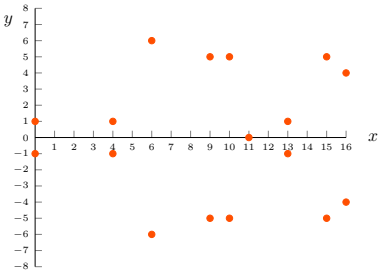
Figure 1.1: Examples of (smooth) elliptic curves over $\mathbb{R}$ and $\mathbb{F}_{17}$ and singular curves over $\mathbb{R}$



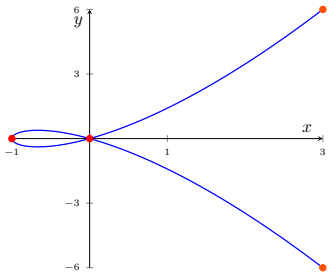
(a) $E/\mathbb{R}$: $y^2 = x^3 - 2$



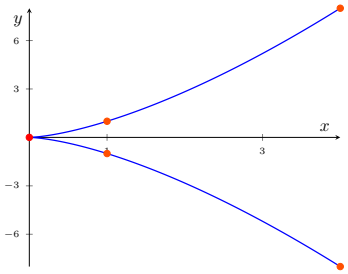
(b) $E/\mathbb{R}$: $y^2 = x^3 - 6x + 4$



(c) $E/\mathbb{F}_{17}$: $y^2 = x^3 + x + 1$



(d) $E/\mathbb{R}$: $y^2 = x^2(x+1)$ (singular: node)



(e) $E/\mathbb{R}$: $y^2 = x^3$(singular: cusp)

(c) Define λ as

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{if } P_1 \neq P_2 \\ (3x_1^2 + a)/(2y_1) & \text{if } P_1 = P_2 \end{cases}$$

The point $P_3 = P_1 + P_2$ is given by $P_3 = (x_3, y_3)$ with

$$x_3 = \lambda^2 - x_1 - x_2; \quad y_3 = \lambda(x_1 - x_3) - y_1 \ .$$

A geometric interpretation. To add P_1 and P_2, one takes the line ℓ passing through them. If the points are equal, one takes the tangent to E in P_1 $(= P_2)$. From Bézout's theorem, we know that ℓ intersects with E in a third point. The reflection of this third intersection point about the x-axis is the sum P_3. Figure 1.2 shows this geometric interpretation of the group law over $\mathbb{R}$.

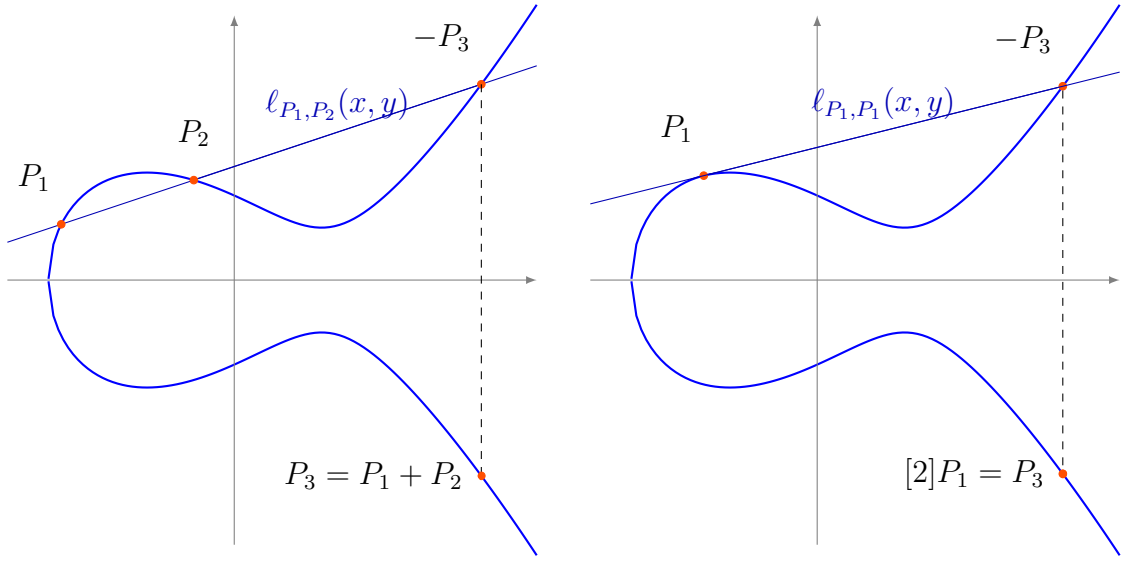
Scalar multiplication. The multiplication-by-m map, or scalar multiplication is

$$\begin{array}{rccc} [m]\colon & E & \to & E \\ & P & \mapsto & \underbrace{P + \ldots + P}_{m \text{ copies of } P} \end{array}$$

for any $m \in \mathbb{Z}$, with $[-m]P = [m](-P)$ and $[0]P = \mathcal{O}$.

Given $m > 0$, computing $[m]P$ as $P + P + \ldots P$ with $m - 1$ additions is **exponential** in the size of m: $m = e^{\ln m}$. We can compute $[m]P$ in $O(\log m)$ operations on E with the naive double-and-add method (Alg. 1.1). Better algorithms can be derived with different scalar encodings (NAF [MO90], DBNS [DIM05]), windowing methods and efficient endomorphisms if available (GLV [GLV01]).

Figure 1.2: Chord-and-tangent rule over $\mathbb{R}$



(a) "Chord": $P_1 + P_2$ (b) "Tangent": $2P$

Algorithm 1.1: Double-and-add algorithm for scalar multiplication
Input: E defined over $\mathbb{F}_p$, $m > 0$, $P \in E(\mathbb{F}_p)$
Output: $[m]P \in E$

if $m = 0$ **then return** $\mathcal{O}$
Write m in binary expansion $m = \sum_{i=0}^{n-1} b_i 2^i$ where $b_i \in \{0, 1\}$
$R \leftarrow P$
for $i = n - 2$ *downto* 0 **do** loop invariant: $R = [\lfloor m/2^i \rfloor]P$
    $R \leftarrow [2]R$
    if $b_i = 1$ **then**
        $R \leftarrow R + P$
return R

1.2 Background on pairings

We recall elementary definitions of pairings and present the computation of two pairings used in practice, the Tate and ate pairings. All elliptic curves discussed below are *ordinary* (i.e. non-supersingular).

Let E be an ordinary elliptic curve defined over $\mathbb{F}_p$. Let π_p be the Frobenius endomorphism: $(x, y) \mapsto (x^p, y^p)$. Its minimal polynomial is $X^2 - tX + p$ where $t \neq 0$ is called the *trace*. Let r be a prime divisor of the curve order $\#E(\mathbb{F}_p) = p + 1 - t$. The r-torsion subgroup of E is denoted $E[r] := \{P \in E(\overline{\mathbb{F}_p}), [r]P = \mathcal{O}\}$ and has two subgroups of order r (eigenspaces of π_p in $E[r]$) that are useful for pairing applications. We define the two groups $\mathbb{G}_1 = E[r] \cap \ker(\pi_p - [1])$ with a generator denoted by G_1, and $\mathbb{G}_2 = E[r] \cap \ker(\pi_p - [p])$ with a generator G_2. The group $\mathbb{G}_2$ is defined over $\mathbb{F}_{p^k}$, where the embedding degree k is the smallest integer $k \in \mathbb{N}^*$ such that $r \mid p^k - 1$.

We recall the Tate and ate pairing definitions, based on the same two steps: evaluating a function $f_{s,Q}$ at a point P, the Miller loop step, and then raising it to the power $(p^k - 1)/r$,

the final exponentiation step. The function $f_{s,Q}$ has divisor $\mathrm{div}(f_{s,Q}) = s(Q) - ([s]Q) - (s-1)(\mathcal{O})$ and satisfies, for integers i and j,

$$f_{i+j,Q} = f_{i,Q} f_{j,Q} \frac{\ell_{[i]Q,[j]Q}}{v_{[i+j]Q}},$$

where $\ell_{[i]Q,[j]Q}$ and $v_{[i+j]Q}$ are the two lines needed to compute $[i+j]Q$ from $[i]Q$ and $[j]Q$ (ℓ intersecting the two points and v the vertical). We compute $f_{s,Q}(P)$ with the Miller loop presented in Algorithm 1.2.

Algorithm 1.2: MillerLoop(s, P, Q)

Output: $m = f_{s,Q}(P)$ for $s = \sum_{i=0}^{t} s_i 2^i$

$m \leftarrow 1$; $S \leftarrow Q$;
for b *from* $t-1$ *to* 0 **do**
  $\ell \leftarrow \ell_{S,S}(P)$; $S \leftarrow [2]S$; // DoubleLine
  $v \leftarrow v_{[2]S}(P)$; // VerticalLine
  $m \leftarrow m^2 \cdot \ell / v$; // Update1
  if $s_b = 1$ **then**
    $\ell \leftarrow \ell_{S,Q}(P)$; $S \leftarrow S + Q$; // AddLine
    $v \leftarrow v_{S+Q}(P)$; // VerticalLine
    $m \leftarrow m \cdot \ell / v$; // Update2
return m;

The Tate and ate pairings are defined by

$$\mathrm{Tate}(P,Q) \coloneqq f_{r,P}(Q)^{(p^k-1)/r}$$
$$\mathrm{ate}(P,Q) \coloneqq f_{t-1,Q}(P)^{(p^k-1)/r}$$

where $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$. The final exponentiation eliminates any factor which lives in a strict subfield of $\mathbb{F}_{p^k}$ [BKLS02]. In case the embedding degree k is even, the vertical lines $v_{S+Q}(P)$ and $v_{[2]S}(P)$ live in a strict subfield of $\mathbb{F}_{p^k}$ so these factors will be neutralized by the final exponentiation. Hence, in this situation we ignore the VerticalLine steps and remove the divisions by v in Update1 and Update2 steps.

In the sequel, when abstraction is needed, we refer to a pairing as the bilinear map (cf. 1.3):

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

Complex multiplication and endomorphisms. It is also important to recall some results with respect to the complex multiplication (CM) discriminant $-D$. When $D = 3$ (resp. $D = 4$), the curve has CM by $\mathbb{Q}(\sqrt{-3})$ (resp. $\mathbb{Q}(\sqrt{-1})$) so that twists of degrees 3 and 6 exist (resp. 4). When E has d-th order twists for some $d \mid k$, then $\mathbb{G}_2$ is isomorphic to $E'[r](\mathbb{F}_{p^{k/d}})$ for some twist E'. Otherwise, in the general case, E admits a single twist (up to isomorphism) and it is of degree 2. We denote c_1 and c_2 the $\mathbb{G}_1$ and resp. $\mathbb{G}_2$ cofactors, *i.e* $\#E(\mathbb{F}_p) = c_1 r$ and $\#E'(\mathbb{F}_{p^{k/d}}) = c_2 r$.

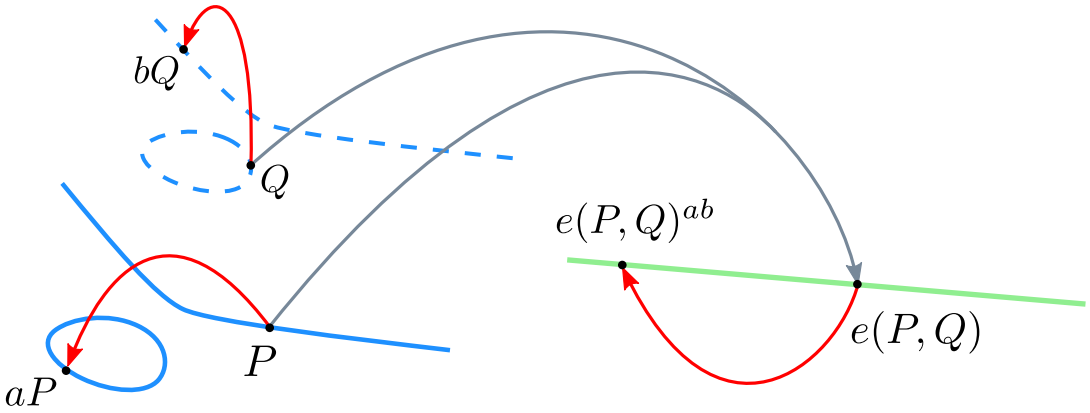


Figure 1.3: A bilinear pairing $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ [*].

[*] Figure courtesy of Diego F. Aranha.

When $D = 3$, the curve has a j-invariant 0 and is of the form $y^2 = x^3 + b$ ($a = 0$). In this case, an efficient endomorphism ϕ exists on $\mathbb{G}_1$. Given β a cube root of unity in $\mathbb{F}_p$,

$$\begin{aligned} \phi : E(\mathbb{F}_p)[r] &\to E(\mathbb{F}_p)[r] \\ (x, y) &\mapsto (\beta x, y) \quad (\text{and } \mathcal{O} \mapsto \mathcal{O}) \ . \end{aligned}$$

The endomorphism ϕ has a minimal polynomial $X^2 + X + 1$ and an eigenvalue λ satisfying $\lambda^2 + \lambda + 1 \equiv 0 \mod r$. When $D = 1$, the curve has j-invariant 1728 and is of the form $y^2 = x^3 + ax$ ($b = 0$). In this case an efficient endomorphism σ exists on $\mathbb{G}_1$. Given $i \in \mathbb{F}_p$ such that $i^2 = -1$,

$$\begin{aligned} \sigma : E(\mathbb{F}_p)[r] &\to E(\mathbb{F}_p)[r] \\ (x, y) &\mapsto (-x, iy) \quad (\text{and } \mathcal{O} \mapsto \mathcal{O}) \ . \end{aligned}$$

On $\mathbb{G}_2$, an efficient endomorphism is ψ the "untwist-Frobenius-twist" introduced in [GS08a]. ψ has a minimal polynomial $X^2 - tX + p$ and is defined by

$$\begin{aligned} \psi : E'[r](\mathbb{F}_{p^{k/d}}) &\to E'[r](\mathbb{F}_{p^{k/d}}) \\ (x, y) &\mapsto \xi^{-1} \circ \pi_p \circ \xi(x, y) \quad (\text{and } \mathcal{O} \mapsto \mathcal{O}) \ . \end{aligned}$$

where ξ is the twisting isomorphism from E' to E. When $D = 3$, there are actually two sextic twists, one with $p + 1 - (-3y + t)/2$ points on it, the other with $p + 1 - (3y + t)/2$, where $y = \sqrt{(4p - t^2)/3}$. Only one of these is the "right" twist, i.e. has an order divisible by r. Let ν be a quadratic and cubic non-residue in $\mathbb{F}_{p^{k/d}}$ and $X^6 - \nu$ an irreducible polynomial, the "right" twist is either $y^2 = x^3 + b/\nu$ (D-type twist) or $y^2 = x^3 + b\nu$ (M-type twist). For the D-type, the twisting isomorphism from E' to E is $\xi : (x, y) \mapsto (\nu^{1/3}x, \nu^{1/2}y)$ and ψ becomes

$$\psi : (x, y) \mapsto (\nu^{(p-1)/3}x^p, \nu^{(p-1)/2}y^p) \quad (\text{and } \mathcal{O} \mapsto \mathcal{O}) \ .$$

For the M-type, $\xi : (x, y) \mapsto (\nu^{2/3}x/\nu, \nu^{1/2}y/\nu)$ and ψ becomes

$$\psi : (x, y) \mapsto (\nu^{(-p+1)/3}x^p, \nu^{(-p+1)/2}y^p) \quad (\text{and } \mathcal{O} \mapsto \mathcal{O}) \ .$$

For other d-twisting ξ formulae, see [Sco09].

Most of pairing-friendly curves fall into polynomial families, *i.e.* the curve parameters are expressed as polynomials $p(x), r(x)$ and $t(x)$. These polynomials are then evaluated at a "seed" u to derive a given curve (cf. Sec. 1.3).

1.3 Some pairing-friendly constructions

We recall some methods and families from the literature related to pairing-friendly constructions that will be of interest in the following sections. A detailed study of these constructions is available in the taxonomy paper by Freeman, Scott and Teske [FST10]. We focus on the Cocks–Pinch [FST10, Sec. 4.1] (CP) and Brezing–Weng [FST10, Sec. 6.1] (BW) methods, and Barreto–Lynn–Scott [FST10, Sec. 6.6] of embedding degrees 12 and 24 (BLS12 and BLS24), Barreto–Naehrig [BN06] (BN), Miyaji–Nakabayashi–Takano [FST10, Sec. 5.1] of embedding degrees 4 and 6 (MNT4 and MNT6), Galbraith–McKee-Valença [GMV04] [FST10, Sec. 5.2] of embedding degree 6 (GMV6) and cofactor 4 families and Kachisa–Schaefer–Scott [KSS08] of embedding degrees 16 and 18 (KSS16 and KSS18) families.

From a high level perspective, there are two ways to obtain pairing-friendly curves. Generic algorithms take as inputs k and r, and output (if it exists) an elliptic curve defined over a field $\mathbb{F}_p$ and of embedding degree k w.r.t. a subgroup of prime order r over $\mathbb{F}_p$. If r is an integer, this is the Cocks–Pinch method, if r is a polynomial, this is the Brezing–Weng method. The alternative is to consider precomputed tables of polynomial families $(k, r(x), D, t(x), p(x))$ as in [FST10]. To rank the families of the same k, the ρ-value is defined as the ratio of the sizes of p and r, resp. the ratio of the degrees of the polynomials $p(x)$ and $r(x)$:

$$\rho = \frac{\log p}{\log r}, \quad \text{resp.} \quad \rho = \frac{\deg p(x)}{\deg r(x)} \tag{1.1}$$

and because $r \mid p + 1 - t$, then $\rho \geq 1$.

Cocks–Pinch is the most flexible method and can be used to construct a curve $E(\mathbb{F}_p)$ with arbitrary embedding degrees and a subgroup order r such that the ratio $\rho = \log_2 p / \log_2 r$ is approximately 2. It works by fixing r and the CM discriminant D and then computing the trace t and the prime p such that the CM equation $4p = t^2 + Dy^2$ (for some $y \in \mathbb{Z}$) is satisfied (cf. Alg. 1.3).

Algorithm 1.3: Cocks–Pinch method
Input: A positive integer k and a positive square-free integer D
Output: $E/\mathbb{F}_p$ with an order-r subgroup and embedding degree k

1 Choose a prime r such that k divides $r - 1$ and $-D$ is a square modulo r;
2 Compute $t = 1 + z^{(r-1)/k}$ for z a generator of $(\mathbb{Z}/r\mathbb{Z})^\times$;
3 Compute $y = (t - 2)/\sqrt{-D} \bmod r$;
4 Lift t and y in $\mathbb{Z}$;
5 Compute $p = (t^2 + Dy^2)/4$ in $\mathbb{Q}$;
6 **if** *p is a prime integer* **then**
7 | Use CM method ($D < 10^{12}$) to construct $E/\mathbb{F}_p$ with order-r subgroup;
8 **else**
9 | Go back to 1;
10 **return** *$E/\mathbb{F}_p$ with an order-r subgroup and embedding degree k*

Barreto, Lynn and Scott [BLS03] and later Brezing and Weng [BW05] generalized the Cocks–Pinch method by parameterizing t, r and p as polynomials. This led to curves with a ratio $\rho < 2$. Below, we sketch the idea of the algorithm in its generality for both BLS and BW constructions (cf. Alg. 1.4). Particular choices of polynomials for $k = 12$ and $k = 24$ yield two families of curves with good security/performance trade-offs, denoted respectively BLS12 and BLS24. The parameters are given in Table 1.1.

Algorithm 1.4: Idea of BLS and BW methods
Input: A positive integer k and a positive square-free integer D
Output: $E/\mathbb{F}_{p(x)}$ with an order-$r(x)$ subgroup and embedding degree k
1 Choose an irreducible polynomial $r(x) \in \mathbb{Z}[x]$ with a positive leading coefficient such that $\sqrt{-D}$ and the primitive k-th root of unity ζ_k are in $K = \mathbb{Q}[x]/(r(x))$;
2 Choose $t(x) \in \mathbb{Q}[x]$ be a polynomial representing $\zeta_k + 1$ in K;
3 Set $y(x) \in \mathbb{Q}[x]$ be a polynomial mapping to $(\zeta_k - 1)/\sqrt{-D}$ in K;
4 Compute $p(x) = (t^2(x) + Dy^2(x))/4$ in $\mathbb{Q}[x]$;
5 **while** $p(x)$ *is not irreducible* **do**
6 | Go back to 1;
7 **return** $E/\mathbb{F}_{p(x)}$ *with an order-*$r(x)$ *subgroup and embedding degree* k

MNT curves, however, have a fixed embedding degree $k \in \{3, 4, 6\}$ and a variable discriminant D. They are also parameterized by polynomials and form a sparse family (cf. Table 1.1) because one is required to solve a generalized Pell equation. Karabina and Teske [KT08] showed that there is a prime-order elliptic curve $E/\mathbb{F}_p$ from the MNT6 family if and only if there is a prime-order elliptic curve $E'/\mathbb{F}_q$ from the MNT4 family such that $\#E(\mathbb{F}_p) = q$ and $\#E'(\mathbb{F}_q) = p$.

GMV curves extend MNT curves with cofactors $2 \leq c_1 \leq 5$. This is achieved by following the MNT strategy and substituting $c_1 \cdot r$ for $\#E(\mathbb{F}_p)$ followed by an explicit analysis of the cases $c_1 = 2, 3, 4$ and 5. Polynomial parameters for GMV with $k = 6$ and $c_1 = 4$ are given in Table 1.1 (the parameters in **bold** are used in Sec. 2.2.1). Later, Le et. al [LMHT18] extended these constructions to any cofactor c_1.

BN curves form a family of prime-order pairing-friendly elliptic curves with $k = 12$ and $D = 3$ (cf. Table 1.1). The construction is based on a result from [GMV07] and a lucky try in which the right-hand side of the CM equation happens to be a constant times a perfect square polynomial. However, it was suggested in [FST10, Example 6.8] that the BN construction can be viewed as a complete family on its own.

Another strategy to build pairing-friendly constructions is to pick random small elements and take their minimal polynomials as the subgroup order polynomial $r(x)$. For well chosen embedding degrees $k = 16$ and $k = 18$, this yields the KSS16 and KSS18 families with $\rho = 5/4$ and $\rho = 4/3$ respectively (cf. Table 1.1).

Family	k	D	ρ	$r(x)$	$p(x)$	$t(x)$
BN	12	-3	1	$36x^4+36x^3+18x^2+6x+1$	$36x^4+36x^3+24x^2+6x+1$	$6x^2+1$
BLS12	12	-3	3/2	x^4-x^2+1	$(x^6-2x^5+2x^3-x+1)/3+x$	$x+1$
BLS24	24	-3	5/4	x^8-x^4+1	$(x^{10}-2x^9+x^8-x^6+2x^5-$ $x^4+x^2+x+1)/3$	$x+1$
MNT4	4	D	1	x^2+1 or x^2+2x+2	x^2+x+1	$-x$ or $x+1$
MNT6	6	D	1	$4x^2\pm 2x+1$	$4x^2+1$	$1\pm 2x$
GMV6	6	D	1	$4x^2+2x+1$ $=\Phi_6(t-1)$	$16x^2+10x+5$	$2x+2$
($c_1=4$)				$28x^2+10x+1$ $=\Phi_6(t-1)/7$	$112x^2+54x+7$	$14x+4$
				$28x^2+18x+3$ $=\Phi_6(t-1)/7$	$112x^2+86x+17$	$14x+6$
				$\mathbf{52x^2+14x+1}$ $=\Phi_6(t-1)/13$	$\mathbf{208x^2+30x+1}$	$\mathbf{-26x-2}$
				$52x^2+38x+7$ $=\Phi_6(t-1)/13$	$208x^2+126x+19$	$-26x-8$
KSS16	16	-1	5/4	$(x^9+48x^4+625)/61550$	$(x^{10}+2x^9+5x^8+48x^6+152x^5+$ $240x^4+625x^2+2398x+3125)/980$	$(2x^5+41x+35)/35$
KSS18	18	-3	4/3	$(x^6+37x^3+343)/343$	$(x^8+5x^7+7x^6+37x^5+188x^4+$ $259x^3+343x^2+1763x+2401)/21$	$(x^4+16x+7)/7$

Table 1.1: Polynomial parameters of BN, BLS12, MNT4, MNT6, GMV6, KSS16 and KSS18 families.

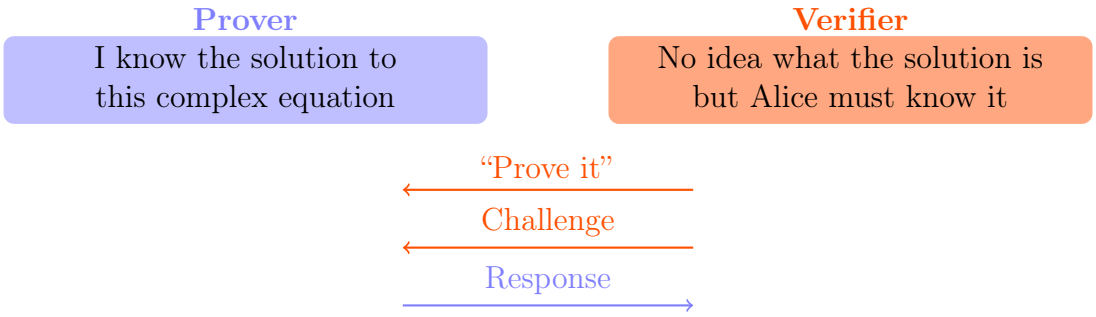
Chapter

2

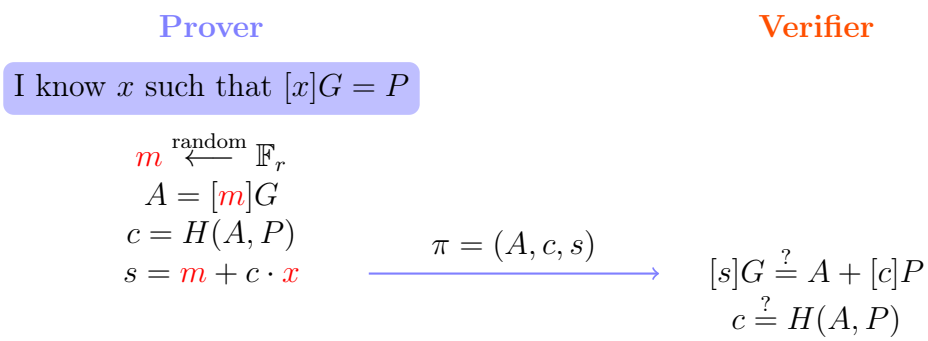
Pairing-based proof systems

2.1 Proof systems

A proof system is an interactive protocol where one party (called the prover) tries to convince another party (called the verifier) that a given statement is true. In a zero-knowledge proof, we further require that the proof does not reveal anything beyond the truth of the statement. A proof is non-interactive if no communication is required between the prover and the verifier except from sending the proof.



Example: sigma protocol



In the class of non-interactive proofs, a particularly interesting concept for proving the computational integrity is the Succinct Non-interactive ARgument of Knowledge (SNARK). It provides a computationally sound proof that is cheap to verify and small compared to the size of the statement or the witness. SNARK systems can be further equipped with a zero-knowledge property that enables the proof to be verified without revealing anything about the intermediate steps (the witness generation). We say that a zk-SNARK is:

- Sound, if No cheating prover can convince an honest verifier of a false statement;
- Complete, if An honest prover always convinces an honest verifier of a true statement;
- Succinct, if An honestly-generated proof is "short" and "easy" to verify;
- Non-interactive, if No interaction between the prover and verifier is required;
- ARgument of Knowledge, if An honest verifier is convinced that a *computationally bounded* prover knows an information related to the statement;
- Zero-knowledge, if a verifier learns nothing other than the truth of the statement.

Proof systems were introduced in [GMR89] and extensively studied both in theoretical and applied settings (e.g. [Kil92, Mic94, GW11, BCCT12]). Recent constructions focus on a panoply of settings that range from cryptographic assumptions, asymptotic efficiency, concrete performance of implementations to numerous applications. The mathematical security of many schemes relies on variants of the discrete logarithm problem (DLP): given a cyclic group $\mathbb{G}$ of prime order r written additively, a generator G, and an element $P \in \mathbb{G}$, compute $x \in \{0, 1, \ldots, r-1\}$ such that $[x]G = P$. For example, the DLP-based zero-knowledge proofs (e.g. [BCC$^+$16], Bulletproofs [BBB$^+$18], Hyrax [WTs$^+$18]) require a group $\mathbb{G}$ where the discrete logarithm problem is hard. They can be instantiated with any cryptographically secure elliptic curve, where the problem is then referred to as the Elliptic Curve Discrete Logarithm Problem (ECDLP). An efficient implementation uses the Ristretto group [Ham15, Val21] over `ed25519` [BDL$^+$12] (e.g. Bulletproofs' Dalek library [dVYA22]).

Alternatively, a bilinear pairing is required in certain schemes. Pairing-based SNARKs cannot be instantiated with general-purpose elliptic curves, but instead require tailored constructions of elliptic curves. More precisely, they need pairing-friendly elliptic curves with additional properties, designed to provide an efficient implementation. In the following chapters, we give an overview of the elliptic curves designed for different proof systems, revisit some constructions and propose some new ones.

zk-SNARK algorithms. In the following, we mainly focus on pairing-based zk-SNARKs for Non-deterministic Polynomial (NP) languages for which we give a basic algorithmic overview. Given a public NP program F, public inputs a and b and a private input w, such that the program F satisfies the relation $F(a, w) \coloneqq b$, a zk-SNARK consists in proving this relation succinctly without revealing the private input w. Given a security parameter λ, it consists of the `Setup`, `Prove` and `Verify` algorithms (cf. 2.1):

$$
\begin{aligned}
(\sigma_p, \sigma_v) &\leftarrow \texttt{Setup}(F, 1^\lambda) \\
\pi &\leftarrow \texttt{Prove}(a, b, w, \sigma_p) \\
0/1 &\leftarrow \texttt{Verify}(a, b, \pi, \sigma_v)
\end{aligned}
$$

where σ_p is the proving key which encodes the program F for the prover, σ_v the verification key which encodes F for the verifier and π the proof. If the `Setup` algorithm is trapdoor-ed an additional secret input τ is required $(\sigma_p, \sigma_v) \leftarrow \texttt{Setup}(F, \tau, 1^\lambda)$.

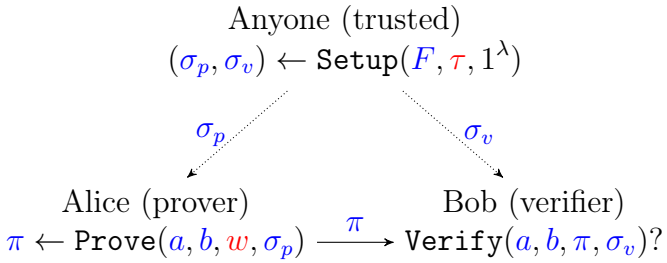


Figure 2.1: zk-SNARK algorithms. Public parameters are in blue and private ones in red.

Definition 2.1 ([BCTV14b]). *A succinct proof π has size $O_\lambda(1)$ and can be verified in time $O_\lambda(|F| + |a| + |b|)$, where $O_\lambda(.)$ is some polynomial in the security parameter λ.*

2.2 Proof systems and pairings

Building on ideas from the pairing-based doubly-homomorphic encryption scheme [BGN05], Groth, Ostrovsky and Sahai [GOS06, Gro06, GS08b] introduced the pairing-based non-interactive zero-knowledge proofs, yielding the first linear-size proofs based on standard assumptions. Groth [Gro10] combined these techniques with ideas from interactive zero-knowledge proofs to give the first constant-size proofs which are based on constructing a set of polynomial equations and using pairings to efficiently verify these equations. This work relies on two new introduced pairing-based cryptographic assumptions, namely the q-computational power Diffie-Hellman (q-CPDH) and the q-power knowledge of Exponent (q-PKE).

The q-PKE assumption. The knowledge of exponent assumption (KEA) says that given G, $[\alpha]G$ it is infeasible to create P, $\hat{P}$ so that $\hat{P} = [\alpha]P$ without knowing a so that $P = [a]G$ and $\hat{P} = [a]([\alpha]G)$. Bellare and Palacio [BP04] extended this to the KEA3 assumption which says that given G, $[x]G$, $[\alpha]G$, $[x][\alpha]G$ it is infeasible to create P, $\hat{P}$ so that $\hat{P} = [\alpha]P$ without knowing a_0, a_1 so that $P = [a_0]G + [a_1]([x]G)$ and $\hat{P} = [a_0]([\alpha]G) + [\alpha x]([a_1]G)$. The q-PKE assumption is a generalization of KEA and KEA3. It says that given $(G, [x]G, \cdots, [x^q]G, [\alpha]G, [\alpha x]G, \cdots, [\alpha x^q]G)$ it is infeasible to create P, $\hat{P}$ so $\hat{P} = [\alpha]P$ without knowing $a_0, \cdots, a_q$ so $P = \sum_{i=0}^{q} [a_i]([x^i]G)$ and $\hat{P} = \sum_{i=0}^{q} [a_i]([\alpha x^i]G)$.

The q-CPDH assumption. The computational Diffie-Hellman (CDH) assumption says that given $G, [\alpha]G, [x]G$ it is infeasible to compute $[\alpha x]G$. The q-computational power Diffie-Hellman assumption is a generalization of the CDH assumption that says given $(G, [x]G, \cdots, [x^q]G, [\alpha]G, [\alpha x]G, \cdots, [\alpha x^q]G)$ except for one missing element $[\alpha x^j]G$, it is hard to compute the missing element.

The q-CPDH assumption is a standard computational intractability assumption but the q-PKE is a so-called knowledge of exponent assumption. Knowledge of exponent

assumptions have been criticized for being unfalsifiable [Nao03] but the use of a non-standard assumption may be unavoidable for statistical zero-knowledge arguments [AF07].

Overview of the [Gro16] construction. Given an instance $\Phi = (a_0, \ldots, a_\ell) \in \mathbb{F}_r^\ell$ of a public NP program F, the SNARK algorithms (cf. Fig. 2.1) are:

- $(\sigma_p, \sigma_v) \leftarrow \texttt{Setup}(F, \tau, 1^\lambda)$ where

$$\sigma_v = (\sigma_{v_{\alpha,\beta}}, \{\sigma_{v_{\pi_i}}\}_{i=0}^{\ell}, \sigma_{v_\gamma}, \sigma_{v_\delta}) \in \mathbb{G}_T \times \mathbb{G}_1^{\ell+1} \times \mathbb{G}_2 \times \mathbb{G}_2$$

- $\pi \leftarrow \texttt{Prove}(\Phi, w, \sigma_p)$ where

$$\pi = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \qquad (O_\lambda(1), \text{Def. 2.1})$$

- $0/1 \leftarrow \texttt{Verify}(\Phi, \pi, \sigma_v)$ where Verify is

$$e(A, B) = \sigma_{v_{\alpha,\beta}} \cdot e(\sigma_{v_x}, \sigma_{v_\gamma}) \cdot e(C, \sigma_{v_\delta}) \qquad (O_\lambda(|\Phi|), \text{Def. 2.1}) \tag{2.1}$$

and $\sigma_{v_x} = \sum_{i=0}^{\ell} [a_i]\sigma_{v_{\pi_i}}$ depends only on the instance Φ and $\sigma_{v_{\alpha,\beta}} = e(\sigma_{v_\alpha}, \sigma_{v_\beta})$ can be computed in the trusted setup for $(\sigma_{v_\alpha}, \sigma_{v_\beta}) \in \mathbb{G}_1 \times \mathbb{G}_2$.

The Setup and Prove algorithms mainly consist in multi-scalar multiplications (MSM) in $\mathbb{G}_1$ and $\mathbb{G}_2$ groups and polynomial arithmetic in $\mathbb{F}_r$, using Fast Fourier Transforms (FFTs) (cf. Table 4.21).

2.2.1 Pairing-friendly curves for SNARKs

Following this direction of work, Gennaro et al. [GGPR13] proposed an insightful construction of polynomial equations that resulted in many interesting implementations [PHGR13, BFR+13, BCG+13, BCTV14b, KPP+14] leading to the most succinct and widely implemented pairing-based SNARK [Gro16]. The first implementation, Pinocchio [PHGR13] used a pairing-friendly elliptic curve in the BN family [BN06] (BN) targeting a 128-bit security level, but the source code was proprietary. They used the BN curve defined over a 256-bit field suggested in [NNS10] (seed $x = 1868033^3$). Next, as part of Pantry [BFR+13], authors re-implemented Pinocchio under a BSD-style license using a 254-bit BN curve from [BGM+10] (seed $x = -(2^{62} + 2^{55} + 1)$, first introduced in [NAS+08]). This new BN implementation partially builds on techniques from the previous BN paper [NNS10] Pinocchio used.

Later in [BCG+13], the authors observed that constructing a pairing-friendly curve with a subgroup order r where $r - 1$ is divisible by 2^L a large power of 2, results in an efficient proof generation via suitable Fast Fourier Transforms (FFTs) in $\mathbb{F}_r$. To speed up the arithmetic, they proposed to use the elliptic curve in Edwards form, by looking for a group order multiple of 4. To match these two constraints: 2^L divides $r - 1$ and the curve has order $4 \cdot r$, they designed a Galbraith-McKee-Valença curve [GMV07] of embedding degree 6 (GMV6) defined over $\mathbb{F}_p$ where p is a prime of 183 bits, and of order $4r$ where r is of 181 bits such that $2^{31} \mid r - 1$. This curve was targeting a 80-bit security level in 2013 and was implemented in libff [BSCT+a].

The same year, [BCTV14b] improved previous SNARK works and developed their implementations with two different curves: the same GMV6 curve for the estimated 80-bit security level ($2^{31} \mid r-1$) and a new BN curve for the estimated 128-bit security level ($2^{28} \mid r-1$). Note that, as of today, this BN curve is the one precompiled in the Ethereum blockchain. Last, Trinocchio [KPP+14] provided an implementation of a privacy-preserving version of Pinocchio using the BN curve from [BCTV14b]. As the number of applications increases, other implementations were released especially in the blockchain space. In particular, Zcash cryptocurrency first implemented Zerocash protocol [BCG+14] which uses the SNARK of [BCTV14b] and its BN curve, before switching to the SNARK of [Gro16] and a new curve.

Security of pairing-based schemes. The previous paragraphs told an overview over ten years of active and fruitful research on SNARK from the prelude [BGN05] in 2005 to 2016 with [Gro16]. Meanwhile, the research in discrete logarithm computation in finite fields related to pairings also saw a tremendous decade, with a prequel in 2012 with a discrete logarithm record-breaking computation in $\mathrm{GF}(3^{6\cdot 97})$ in [FNK12]. This announcement had a quite important impact over the community at that time [GM16, §9.3.8 p.30], probably because the broken curve was introduced with the BLS short signatures [BLS01]. The targeted finite field $\mathrm{GF}(3^{6\cdot 97})$ was considered safe for 80-bit security implementations in 2012. In 2014, two algorithms on fast computation of DL in small characteristic finite fields were published [BGJT14, GKZ14] (quasi-polynomial-time algorithm, zig-zag descent). In 2019, Granger et al. announced the largest record computation in a field $\mathrm{GF}(2^{30750})$ [GKL+21], and finally [KW22] published a proved complexity. Nowadays, small-characteristic finite fields should be definitely avoided, as computing DL is not hard anymore especially if the extension degree is composite, which renders supersingular elliptic curves in small characteristic insecure.

Cryptanalysts also considered large and medium characteristic finite fields of the form $\mathrm{GF}(p^k)$ that arise with pairings. In 2016, Kim and Barbulescu [KB16] presented a variant of the Number Field Sieve (NFS) algorithm which reduced the security level of the BN curves previously at 128-bit security to around 110. Menezes, Sarkar and Singh [MSS16] were the first to analyze thoroughly the consequences of the new NFS variants on the security of pairing-friendly curves. Their conclusions recommend the Barreto–Lynn–Scott curves of embedding degree 12 [BLS04] or BN curves over 384-bit prime fields instead of BN curves over 256-bit fields. Based on this work, the Zcash team derived the BLS12-381 curve defined over a 381-bit field with a 255-bit prime subgroup order r such that $2^{32} \mid r-1$ [Bow17]. This curve is used today in several projects (e.g. Zcash, Ethereum 2.0, Skale, Algorand, Dfinity, Chia), implemented in different libraries and considered for IETF standards.

2.2.2 Pairing-friendly curves for recursive SNARKs

Besides their efficiency, SNARKs' succinctness makes them good candidates for recursive proof composition. Such proofs could themselves verify the correctness of other proofs. This would allow a single proof to inductively attest to the correctness of many former proofs. However, once a first proof is generated, it is highly impractical to use the same elliptic curve to generate a second proof verifying the first one. In pairing-based SNARKs the proving algorithm runs in $\mathbb{F}_r$ while the verification algorithm runs in $\mathbb{F}_p$. Ideally, we would like to select a curve E with $p = r$, so that proving arithmetic is over the same field

for which the statement (the verification of the previous proof) is defined. Unfortunately, this cannot happen: the condition that E has embedding degree k with respect to r implies that $r \mid p^k - 1$, which implies that $p \neq r$. Furthermore, the curves satisfying this condition have a trivial discrete logarithm problem. So, while appealing, this idea cannot even be instantiated. Since we are stuck with $p \neq r$, we may consider doing "long arithmetic": simulating $\mathbb{F}_p$ operations via $\mathbb{F}_r$ operations, by working with bit chunks to perform integer arithmetic, and modding out by p when needed. Alas, having to work at the "bit level" implies a blowup on the order of $\log p$ compared to native arithmetic. So, while this approach can at least be instantiated, it is very expensive. A practical approach requires two different curves $E(\mathbb{F}_p)$ and $E'(\mathbb{F}_q)$ that are closely tied together (cf. Fig 2.2).

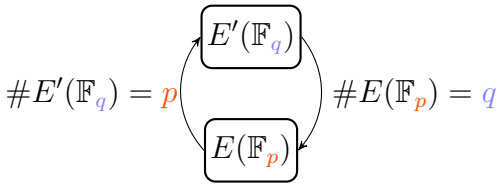


Figure 2.2: A cycle of elliptic curves.

Therefore, we need to find new pairing-friendly curve constructions for this problem. In practice, given a curve equation over a finite field, computing its order can be done efficiently with the SEA algorithm, while given p and r, computing the parameters of the curve equation involves computing a class polynomial (e.g. with Sutherland software [Sut11, ES10]), and it is infeasible if the curve discriminant D is too large (say more than 20 decimal digits). Ben–Sasson et al. [BCTV14a] presented the first practical setting of a recursive proof composition with a cycle of two MNT pairing-friendly elliptic curves [MNT01]. Proofs generated from one curve can feasibly reason about proofs generated from the other curve resulting in an unbounded number of recusion layers. To achieve this, one curve's order is the other curve's base field order and vice-versa (i.e. $\#E'(\mathbb{F}_q) = p$ and $\#E(\mathbb{F}_p) = q$). The two curves are necessarily of prime order [CCW19, Sec. 7], hence cannot admit an Edwards form. Moreover, they have low embedding degrees (4 and 6) resulting in large base fields to achieve a standard security level. The authors proposed a pair of MNT curves with parameters of 298 bits which they estimated to meet the 80-bit security level in 2014. It should be noted that it was Karabina and Teske [KT08] who first showed in 2008 that there always exist a MNT4/MNT6 cycle.

Around approximately the same time as [BCTV14a], Costello et al. [CFH+15] built on this idea to obtain a bounded recursive proof composition using a 2-chain of two elliptic curves (cf. Fig 2.3), i.e. a BN curve (with seed $x = -(2^{62} + 2^{55} + 1)$ from [NAS+08]) defined over a 254-bit field $\mathbb{F}_p$ and a BW6-509 curve constructed using the Brezing–Weng method (BW) in a way that it has a prime subgroup order equal to p the field characteristic of BN's $\mathbb{F}_p$. This set of curves allows a one-layer recursion. Note that both MNT4 and MNT6 respective $\mathbb{F}_r^*$ fields contain large enough powers of two ($2^{17} \mid r - 1$) while Geppetto's BN and BW6 are not. Later on, [BCTV14a] authors found a new MNT4/6 pair with parameters of 753 bits which they estimated at the 128-bit security level. They shared the parameters of this latter pair with the Coda protocol [BMRS20] developers (now Mina) who used it to build a recursive SNARK-based light blockchain. They updated the preprint with this pair of parameters only recently (2020).

A few years after [BCTV14a], motivated by a new proof composition application (decentralized private computation), ZEXE [BCG+20] proposed a new chain of curves. It

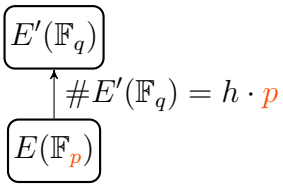


Figure 2.3: A 2-chain of elliptic curves.

is similar to Geppetto's chain although the citation was missing. The first curve is a BLS12-377 defined over a 377-bit field with a subgroup r of 253 bits ($2^{47} \mid r-1$) and, using the more rudimentary Cocks-Pinch method (CP), the second curve is a CP6-782 of embedding degree 6, defined over a 782-bit field and having a group order divisible by BLS12-377's base field size p. Based on this inner BLS12-377, we proposed in [EHG20] an alternative outer curve to CP6-782 that is much faster for a recursive SNARK [Gro16] generation and verification. We used a variant of the Brezing–Weng method to construct a BW6-761 curve of embedding degree 6 defined over a 761-bit field and enjoying properties for efficient implementation. Later, in [EHG22], we generalized the BW6-761 curve construction, the pairing formulas and the group operations to any BW6 curve defined on top of any BLS12 curve, forming a family of 2-chain pairing-friendly curves.

Universal zk-SNARKs Up to this point, all the pairing-based SNARKs use the three pairing groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ whether for proving, verifying or setting the SNARK up. Therefore, all previously mentioned curves were constructed in order to optimize operations in the three different groups. Note that the most efficient pairing-based SNARKs have a trapdoor-ed setup phase specific to the statement to prove (e.g. [Gro16]). Recently, a new kind of SNARKs was introduced, where the setup phase is not specific to a given statement but is rather universal in that sense. Groth et al. [GKM+18] proposed a universal SNARK with a single setup to prove all statements of a given bounded size. However, Sonic [MBKM19] is considered to be the first practical universal SNARK. This work inspired many researchers and practitioners who then came up with new and elegant universal constructions (e.g. AuroraLight [Gab19], PLONK [GWC19], Marlin [CHM+20]).

These universal constructions follow a similar design as Sonic and they use as a fundamental building bloc a **polynomial commitment** (PC) scheme. While there are different PC schemes with trade-offs, the KZG scheme [KZG10] remains the most efficient. It uses pairings and therefore its implementation requires a suitable pairing-friendly elliptic curve. Contrary to the previous pairing-friendly curves used in the SNARK context, KZG-based universal SNARKs need a curve optimized only for $\mathbb{G}_1$ arithmetic and pairings. In fact, on the one hand, in pairing-based SNARKs with a circuit-specific setup phase, the pairing is used to verify that some polynomial identities hold in a secret point included in the trapdoored setup. In KZG-based universal, on the other hand, the pairing is used to open a polynomial commitment (and element in $\mathbb{G}_1$) to a field element, and the polynomial identities are verified in the field. This observation inspired us in [EHG22], to investigate the use of Barreto–Lynn–Scott curves of embedding degree 24 (BLS24) to instantiate KZG-based universal SNARKs. At the 128-bit security level, the coefficient ring of the elements of BLS24 $\mathbb{G}_1$ is much smaller compared to BLS12. We proposed a BLS24-315 curve defined over a 315-bit field with a subgroup order r of 253 bits such that $2^{22} \mid r-1$. Moreover, similarly to the BLS12-BW6 chains, we characterized all 2-chains that can be formed with a BLS24 as an inner curve and BW6 as an outer curve. We short-listed a

few BW6 curves (e.g. BW6-633 and BW6-672) and looked into Cocks-Pinch curves of higher embedding degrees (CP8 and CP12) for a more conservative security. This work was implemented in the `gnark` ecosystem [BPH+22a] using PLONK proof system.

Overview of the KZG polynomial commitment. A polynomial commitment scheme allows to commit to a polynomial and then open it at any point, showing that the value of the polynomial at a point is equal to a claimed value $p(z) = y$. It consists in four algorithms

- $(\sigma_v, \{\sigma_p\}) \leftarrow \texttt{Setup}(\tau, 1^\lambda)$: for some security parameter λ, sample randomly $\tau \leftarrow \mathbb{F}_r$ and then compute $\sigma_p = \tau^i G_1$ for $i \in \{1, \dots, m\}$ and $\sigma_v = \tau G_2$.
- $C \leftarrow \texttt{Commit}(\sigma_p, p)$: given the polynomial $p(x) = \sum_{i=0}^{n} p_i x^i \in \mathbb{F}_r[x]$ of degree $n < m$ and the proving key σ_p, compute $C = p(\tau) \cdot G_1 = \sum_{i=0}^{n<m} p_i \cdot \tau^i G_1$
- $\pi \leftarrow \texttt{Open}(p, y, z)$: to prove that $p(z) = y$, compute the polynomial $q(x) = (p(x) - y)/(x - z)$ and then the proof $\pi = q(\tau) \cdot G_1$
- $0/1 \leftarrow \texttt{Verify}(C, \pi, \sigma_v)$: compute $P_z = zG_2$ and $P_y = yG_1$, and then verify that $e(\pi, \sigma_v - P_z) = e(C - P_y, G_2)$.

Proof (correctness of the KZG protocol).

$$
\begin{aligned}
e(\pi, \sigma_v - P_z) &\equiv e(C - P_y, G_2) \\
e(q(\tau) \cdot G_1, \tau G_2 - zG_2) &\equiv e(p(\tau) \cdot G_1 - yG_1, G_2) \\
\underbrace{e(G_1, G_2)}_{\in \mathbb{G}_T \setminus \{1\}}^{q(\tau)(\tau - z)} &\equiv \underbrace{e(G_1, G_2)}_{\in \mathbb{G}_T \setminus \{1\}}^{p(\tau) - y}
\end{aligned}
$$

□

non-pairing recursion. The universality of these constructions comes from the nature of the polynomial commitment scheme. By swapping the KZG scheme for other PC schemes [BCC+16, VP19, BFS20], one gets new transparent (no setup) SNARKs [KPV19, BFS20] and new transparent recursive SNARKs as first proposed by Bowe, Grigg and Hopwood in [BGH19] (Halo) and then formalized and generalized in [BCMS20, BDFG21]. These recursive constructions build on the discrete logarithm based PC from [BCC+16] and Bulletproofs [BBB+18] and one might want to instantiate them with a non-pairing-friendly elliptic curve like `ed25519`. However, having an efficient recursion requires a **cycle** of elliptic curves as in [BCTV14a], hence the curves are of prime order, which means Edwards and Montgomery forms are not possible. Although this time, the curves do not need to be pairing-friendly. To this end, Halo's authors derived an efficient cycle for SNARK implementation, namely the Tweedledum-Tweedledee cycle. Later, Hopwood proposed the more efficient Pasta cycle [Hop20]. Note that finding such cycles is much easier than finding pairing-friendly cycles. It was investigated previously in a different context by Stange and Silverman [SS11]. It should be noted that it was François Morain who first experimentally discovered the existence of non-pairing cycles in his implementation of ECPP [Mor07]

and their definition was later formalized in [Mih07a] by Preda Mihailscu in the context of primality testing. Finally, as a future work, Halo suggests using the same SNARK techniques with a hybrid cycle where one curve is pairing-friendly and the other is not. Thereafter, Hopwood proposed a hybrid cycle, Pluto-Eris [Hop21]. Such a cycle can be constructed from any prime-order pairing-friendly curve (e.g. BN [BN06], Freeman [Fre10], MNT [MNT01]).

2.2.3 Elliptic curves in SNARKs

SNARKs enable verifying non-deterministic polynomial time (NP) computations with substantially lower complexity than those required for classical NP verification. In many applications (e.g. privacy-preserving cryptocurrencies, zk-rollups, verifiable computation outsourcing), the NP-computation revolves around proving the knowledge of a hash preimage or the verification of a cryptographic signature. In CØCØ [KZM+15], Kosba et al. proposed a library of cryptographic primitives that can be efficiently proved in a SNARK. In particular, the authors looked into proving an Elliptic Curve Diffie-Hellman (ECDH) key exchange. They constructed a new elliptic curve to efficiently implement the operation required in key exchanges, i.e. the scalar multiplication (cf. Fig. 2.4).

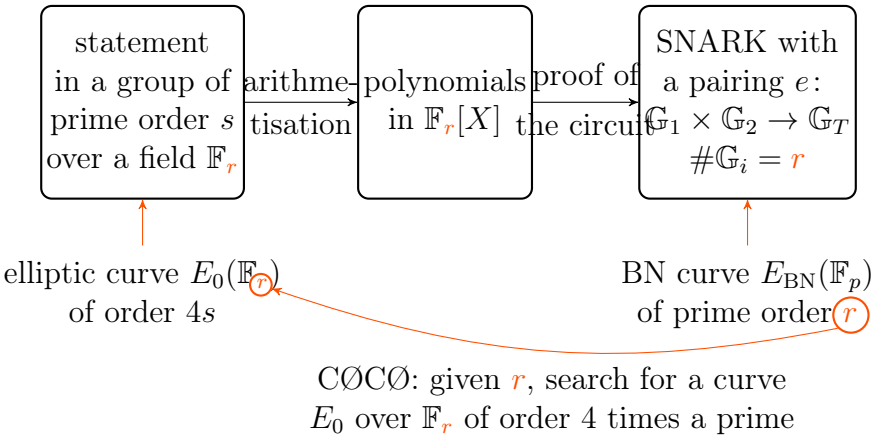


Figure 2.4: Kosba et al. construction [KZM+15]

This curve defined over the scalar field $\mathbb{F}_r$ of the BN curve, with the seed `0x44e992b44a6909f1` from [BCTV14a], was given in Montgomery form for further optimizing the arithmetic inside a SNARK. The paper also mentioned converting this associated curve to Edwards form in order to efficiently prove EdDSA signatures. Following this work, the Zcash team introduced the JubJub curve [ZCa21] which is a similar curve in twisted Edwards form associated to BLS12-381, alongside further algebraic optimizations. This curve allowed Zcash to efficiently implement a collision-resistant variant of an EC-based Pedersen hash inside a SNARK. Practitioners who proposed tailored elliptic curves for SNARKs (BN254, BLS12-377, BLS24-315, BW6-761, BW6-633) each time also proposed an associated twisted Edwards curve defined over the scalar field. Finally, Masson et al. [MSZ21] performed an exhaustive search of associated curves over BLS12-381 with small Complex Multiplication discriminant in order to speed up the curve arithmetic using a fast endomorphism. They found an isolated curve with discriminant $D = -8$, called Bandersnatch. A similar associated curve is unlikely to be found for other SNARK curves of interest.

We stress that the problem solved in the CØCØ construction is partway similar to

the one solved in the Geppetto construction. Remember that in a proof composition, one requires writing efficiently a pairing computation as a SNARK statement and to this end, researchers came up with new elliptic curves that efficiently encode the pairing in the SNARK (cf. Fig. 2.5). In CØCØ, the SNARK curve is fixed ($E_{\mathrm{BN}}(\mathbb{F}_p)$ of order r) and the authors look for a curve ($E_0(\mathbb{F}_r)$) for the statement (ECDH). In Geppetto, the statement is "the verification of a previously generated proof" and its curve is fixed ($E_{\mathrm{BN}}(\mathbb{F}_p)$ of order r) as it was already used to generate that previous proof. The authors look for a new SNARK curve (BW6($\mathbb{F}_s$) of order $h \cdot p$) to prove "the proof composition" statement.

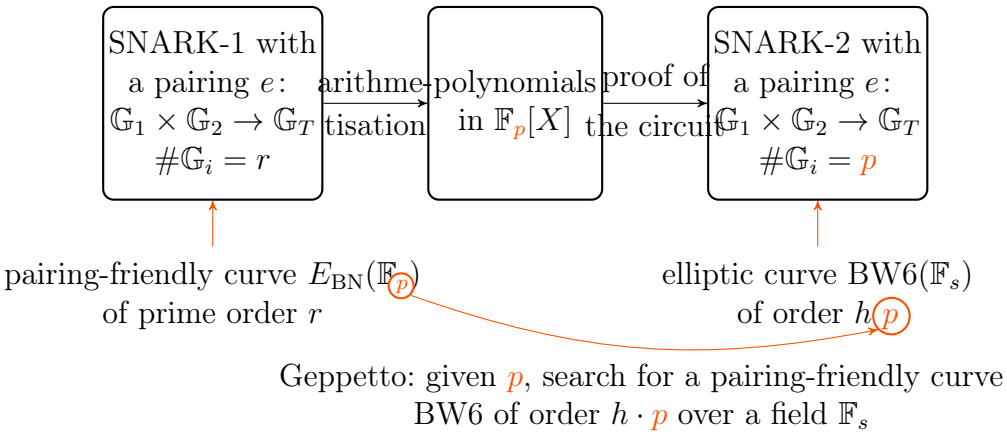


Figure 2.5: Geppetto construction [CFH+15]

II

SNARK-friendly elliptic curves

Chapter

3

Elliptic curves for SNARKs

In this chapter we investigate the use of elliptic curves in pairing-based SNARKs. We derive new tailored constructions and focus on efficient arithmetic, particularly co-factor clearing and subgroup membership testing. This chapter, in part, is a reprint of the material as it appears in our published works [AEHG22] and [EHGP22].

3.1 Constructions

Pairing-based SNARKs can be instantiated with any secure pairing-friendly elliptic curve. For efficiency, we require a fast arithmetic in $\mathbb{F}_r$ (where r is the curve prime subgroup order), in $\mathbb{G}_1$ and in $\mathbb{G}_2$, and a fast pairing computation. For security, we are interested in the 128-bit security level in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$.

3.1.1 Efficiency

The `Setup` and `Prove` algorithms (Fig. 2.1) involve solving multiple large instances of tasks about polynomial arithmetic in $\mathbb{F}_r[X]$ and multi-scalar multiplication (MSM) over $\mathbb{G}_1$ and $\mathbb{G}_2$. The `Verify` algorithm (Fig. 2.1) involves computing a product of pairings and an evaluation in $\mathbb{G}_T$. Fast arithmetic in $\mathbb{F}_r[X]$, when manipulating large-degree polynomials, is best implemented using the Fast Fourier Transform (FFT) [Pol71] and MSMs of large sizes are best implemented using a variant of Pippenger's algorithm [BDLO12, Section 4] (cf. Chapter 5). For example, Table 4.21 reports the numbers of group operations (MSM) required in the `Setup`, `Prove` and `Verify` algorithms in the [Gro16] SNARK and the KZG-based PLONK universal SNARK [GWC19]. The report excludes the number of FFTs as it changes from one implementation to another, but these usually consume less compute time than group operations.

An efficient implementation of the FFT over $\mathbb{F}_r$ requires $r-1$ to split into small primes or ideally to be divisible by a large power of 2. This narrows the search of pairing-friendly elliptic curves to the ones with a subgroup order r such that $2^L \mid r-1$ for a large integer

Table 3.1: Cost of `Setup`, `Prove` and `Verify` algorithms for [Gro16] and PLONK. m = number of wires, n = number of multiplication gates, a = number of addition gates and ℓ = number of public inputs. $\mathtt{M}_{\mathbb{G}}$ = multiplication in $\mathbb{G}$ and P=pairing. *Note:* Both Groth16 and PLONK verifiers have a dependency on the number of public inputs ℓ, but for PLONK it is just a polynomial evaluation (FFT).

	`Setup`	`Prove`	`Verify`
[Gro16]	$3n$ $\mathtt{M}_{\mathbb{G}_1}$ m $\mathtt{M}_{\mathbb{G}_2}$	$(3n+m-\ell)$ $\mathtt{M}_{\mathbb{G}_1}$ n $\mathtt{M}_{\mathbb{G}_2}$	3 P ℓ $\mathtt{M}_{\mathbb{G}_1}$
PLONK (KZG)	$d_{\geq n+a}$ $\mathtt{M}_{\mathbb{G}_1}$ 1 $\mathtt{M}_{\mathbb{G}_2}$	$9(n+a)$ $\mathtt{M}_{\mathbb{G}_1}$	2 P 18 $\mathtt{M}_{\mathbb{G}_1}$

$L \geq 1$. This was suggested first in [BCG$^+$13] and yielded the GMV6-183 curve with $2^{31} \mid r-1$.

Recently, a new algorithm was proposed by Ben–Sasson et al. [BCKL21] that implements a variant of the FFT over non-smooth arbitrary fields $\mathbb{F}_{p'}$ (ECFFT). However, this algorithm suffers from two major drawbacks: finding an elliptic curve over $\mathbb{F}_{p'}$ with a large 2^L-torsion and efficiently implementing the algorithm in the canonical basis. Since the new algorithm is asymptotically much slower than the basic FFT over smooth fields, designing a SNARK curve with smooth $r-1$ remains the best approach.

This narrows the search to an elliptic curve with the following requirements:

(i) valid parameters: integers p, r, t ($p(u), r(u) \in \mathbb{N}$, $t(u) \in \mathbb{Z}$), prime p.

(ii) a subgroup order r such that $2^L \mid r-1$ for a large integer $L \geq 1$,

(iii) a fast pairing,

(iv) a fast arithmetic in $\mathbb{G}_1$ and

(v) a fast arithmetic in $\mathbb{G}_2$.

For KZG-based universal SNARKs, the last requirement can be dropped.

Elliptic curves proposed in the literature to suit SNARKs belong to one of the families discussed in the preliminaries in section 1.3. Except for the Cocks-Pinch curve CP6-782, all the curves are in families defined by polynomials. Condition (ii) becomes a congruence condition on the seed x of the polynomial $r(x)$ (cf. Table 3.2).

Pairing computations take place mainly in $\mathbb{F}_{p^k}$ and it is important to construct the $\mathbb{F}_{p^k}$ tower such that the arithmetic is as efficient as possible. Pairing-friendly tower extensions are built using a sequence of quadratic and cubic sub-extensions. The ideal way is to start with an optimal quadratic extension $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2+1)$ that arises when $p \equiv 3 \bmod 4$. For the discussed families, satisfying condition (iii) boils down then to satisfying a congruence equation in the seed x alongside the previous congruence condition (cf. Table 3.2). Note that even seeds allow an additional speedup in the pairing computation (final exponentiation) for some families (e.g. BLS12 and BLS24 [GF16]). Finally, as is the custom in pairing-based cryptography, the Miller loop scalar (e.g. $u = t-1$ for BLS curves) should have a small Hamming weight. For the curve families we are interested in, the optimal Miller loop scalar is a low-degree polynomial in the seed x, hence we are additionally looking for a sparse seed x in (signed) binary representation.

Condition (iv) is mainly related to the bitsize of p as the point coordinates are in $\mathbb{F}_p$. The bitsize of p varies for each family and is constrained by the security. One requires r of

about 256 bits, and each family has a fixed parameter $\rho = \log_2 p / \log_2 r$, $1 \leq \rho \leq 2$, hence p is usually (but not always) larger than r.

Except for KZG-based universal SNARKs, one requires also a fast arithmetic in $\mathbb{G}_2$. As discussed in the preliminaries, the curves from Table 1.1 have a twist of degree $d = 2$, 4 or 6 and thus $\mathbb{G}_2$ is isomorphic to $E'[r](\mathbb{F}_{p^{k/d}})$ for an appropriate d-twist E'. Condition (v) is immediately related to the choice of k and d (cf. Tab 3.2).

Congruence conditions on the seed x to achieve (ii).

The subgroup order r is given by an irreducible polynomial $r(x)$ in $\mathbb{Q}[x]$. Computing the congruence conditions on x_0 such that $2^L \mid r(x_0) - 1$ given L is equivalent to finding the roots x_0 of $r(x) - 1$ modulo 2^L. Because 2 is a prime, we define in Alg. 3.1 a Hensel-like root-lifting technique inspired by [GS21, §A], with auxiliary functions in Algs. 3.2 and 3.3.

For BN curves, one has $r(x) - 1 = 6x(6x^3 + 6x^2 + 3x + 1)$. With $2^{L-1} \mid x$, then immediately $2^L \mid r - 1$. The other option is $2^{L-1} \mid 6x^3 + 6x^2 + 3x + 1$ and we run Alg. 3.1 (we were not able to derive a generic formula). Note that $\gcd(p(x) - 1, r(x) - 1) = 6x$. For BLS12 and BLS24 curves, $\gcd(p(x) - 1, r(x) - 1) = x - 1$. For BLS12, $r(x) - 1 = x^2(x^2 - 1)$ and for BLS24, $r(x) - 1 = x^4(x^4 - 1)$ and the results of Table 3.2 follow. For KSS16, we did not obtain a generic formula and we derived the condition for $L = 64$ in Table 3.2 with Alg. 3.1. Note that for curves with $D = 1$, $p \equiv 1 \bmod 4$ is required otherwise the curve would be supersingular. For KSS18 curves, Alg. 3.1 outputs another congruence but p is always even in that case so we discarded it. For BN and KSS16 we picked $L = 64$ and obtained the conditions $x_0 \equiv u_0 \bmod c \cdot 2^{L-1}$ but for a smaller $L_0 < L$, the reduction $u_0 \bmod c \cdot 2^{L_0-1}$ gives the answer. Algorithms 3.1, 3.2 and 3.3 are implemented in SageMath at `https://gitlab.inria.fr/tnfs-alpha/alpha`, in the file `sage/tnfs/gen/generate_curve_utils.py`.

3.1.2 Security

We look at the security of all the pairing-friendly families of the proposed curves both from a generic point of view (TNFS attack) and a SNARK-specific point of view (Cheon's attack).

TNFS attack.

In 2015, [BGK15] revisited the Tower-NFS algorithm (TNFS) to compute discrete logarithms in $\mathbb{F}_{p^k}$. Then in 2016, Kim with Barbulescu combined it with other variants of NFS and exploited the extension fields to improve the TNFS algorithm. This resulted in an expected asymptotic complexity $L_Q(\alpha, c) = \exp\big((c + o(1))(\log Q)^\alpha (\log\log Q)^{1-\alpha}\big)$ to be $L_{p^k}(1/3, (48/9)^{1/3} \approx 1.747)$ instead of $L_{p^k}(1/3, (96/9)^{1/3} \approx 2.201)$ with the NFS-HD algorithm of [BGGM15]. More important, the complexity of TNFS is lower than the general NFS algorithm in prime fields: $L_{p^k}(1/3, (64/9)^{1/3} \approx 1.923)$. The key-sizes should be enlarged, and several papers deal with security estimates of TNFS [MSS16, BD19, GS21, Gui20, DGP20]. However these papers *estimate* the security level, but they do not scale with respect to a record computation. This is not yet possible as the first record computation with the TNFS algorithm was published after them, in [DGP21]: it runs the TNFS

[1] For GMV6 and KSS18, one needs to construct respectively the $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{18}}$ towers starting from $\mathbb{F}_{p^3}$. In this case, one looks for the smaller cubic non-residue in $\mathbb{F}_p$.

Family, (i) $r, p \in \mathbb{N},\ t \in \mathbb{Z}$	(ii) $r \equiv 1 \bmod 2^L$	(iii) $p \equiv 3$ mod 4	(v) $\mathbb{G}_2$ coord. in
BN any x	$x \equiv 2570880382155688433 \bmod 2^{63} \Rightarrow 2^{64} \mid r-1$ $x \equiv 0 \bmod 2^{L-1} \Rightarrow 2^L \mid r-1,\ 2^L \mid p-1$	✓ ✗	$\mathbb{F}_{p^2}$
BLS12 $x \equiv 1$ mod 3	$x \equiv 1 \bmod 3 \cdot 2^{L-1} \Rightarrow 2^L \mid r-1,\ 2^{L-1} \mid p-1$ $x \equiv 2^{L-1}-1 \bmod 3 \cdot 2^{L-1} \Rightarrow 2^L \mid r-1,\ 6 \mid p-1$ $x \equiv 2^{L/2} \bmod 3 \cdot 2^{L/2} \Rightarrow 2^L \mid r-1,\ 6 \mid p-1$	✗ ✓ ✓	$\mathbb{F}_{p^2}$
BLS24 $x \equiv 1$ mod 3	$x \equiv 1 \bmod 3 \cdot 2^{L-2} \Rightarrow 2^L \mid r-1,\ 2^{L-2} \mid p-1$ $x \equiv 2^{L-1}-1 \bmod 3 \cdot 2^{L-2} \Rightarrow 2^L \mid r-1,\ 6 \mid p-1$ $x \equiv 2^{L/4} \bmod 3 \cdot 2^{L/4} \Rightarrow 2^L \mid r-1,\ 6 \mid p-1$	✗ ✓ ✓	$\mathbb{F}_{p^4}$
MNT4, $t = x+1$	$x \equiv 0 \bmod 2^{L/2} \Rightarrow 2^L \mid r-1, 2^{L/2} \mid p-1$	✗	$\mathbb{F}_{p^2}$
MNT6	$x \equiv 0 \bmod 2^{L-1} \Rightarrow 2^L \mid r-1, 2^{2L} \mid p-1$	✗	$\mathbb{F}_{p^3}$
GMV6($h=4$) any x	$x \equiv 0 \bmod 2^{L-1} \Rightarrow 2^L \mid r-1,\ 2^{L-1} \mid p-1$	NA[1]	$\mathbb{F}_{p^3}$
KSS16 ($x \equiv \pm 25 \bmod 70$)	$\pm 1439816520986421885, \pm 37456616613123361405$ $\bmod 35 \cdot 2^{62} \Rightarrow 2^{64} \mid r-1,\ p \equiv 1 \bmod 4$	✗	$\mathbb{F}_{p^4}$
KSS18 ($x \equiv 14 \bmod 42$)	$x = 14 \cdot 2^{L/3} \bmod 42 \cdot 2^{L/3} \Rightarrow 2^L \mid r-1,\ 12 \mid p-1$	NA[1]	$\mathbb{F}_{p^3}$

Table 3.2: Conditions (i), (ii), (iii), and (v) for Table 1.1 families. For BN curves with $p \equiv 3 \bmod 4$ and KSS16 curves, it was not possible to obtain a general rule. The residue of $x \bmod 2^L$ is computed by Alg. 3.1 with input $L = 64$ but any L can be given. For KSS18 curves, the other residues x do not give a prime p. Condition (iii) is not possible.

Algorithm 3.1: Congruence conditions on the seed x to achieve (ii)
Input: polynomial $s(x) \in \mathbb{Q}[x]$, modulus m and congruence conditions $\{a_i\}_{i\geq 0}$ such that $x_0 \equiv a_i \bmod m \Rightarrow s(x_0) \in \mathbb{Z}$, prime integer ℓ, integer $L > 0$
Output: Residues u_j, integers L_j and moduli m_j s.t. for all $x_j \equiv u_j \bmod m_j$, $s(x_j) \in \mathbb{Z}$ and $s(x_j) \equiv 0 \bmod \ell^{L_j}$, $L_j \geq L$

for $a_i \in \{a_i\}_{i\geq 0}$**:**
  $s_i(x) \leftarrow s(m \cdot x + a_i) \in \mathbb{Z}[x]$ (this ensure that $s_i(x)$ has integer coefficients)
  $v_i \leftarrow \text{valuation}_\ell(\text{content}(s_i(x)))$
  $s_i(x) \leftarrow s_i(x)/\ell^{v_i}$
  for $r_j \in \mathbb{Z}/\ell\mathbb{Z}$ *a simple root of* $s_i(x)$ *modulo* ℓ**:**
    $r_j \leftarrow \text{lift}_{\mathbb{N}}(r_j)$
    $(u_j, m_j, L_j) \leftarrow$ `lift_simple_root`$(s_i, r_j, a_i + r_j \cdot m, m \cdot \ell, 1 + v_i, \ell, L)$
    $S \leftarrow S \cup \{(u_j, m_j, L_j)\}$
  for $r_j \in \mathbb{Z}/\ell\mathbb{Z}$ *a multiple root of* $s_i(x)$ *modulo* ℓ**:**
    $r_j \leftarrow \text{lift}_{\mathbb{N}}(r_j)$
    $S_j \leftarrow$ `lift_multiple_root`$(s_i, r_j, a_i + r_j \cdot m, m \cdot \ell, 1 + v_i, \ell, L)$
    $S \leftarrow S \cup S_j$
return S

Algorithm 3.2: Hensel lifting of simple roots
Input: polynomial $s_i(x) \in \mathbb{Z}[x]$, modulus m_i, residue $u_i \bmod m_i$, root $r_i \in \mathbb{N}$ s.t. $s_i(r_i) \equiv 0 \bmod \ell$, prime integer ℓ, integers L_i and bound $L > 0$
Output: Residue u_j, integer L_j and modulus m_j s.t. for all $x_j \equiv u_j \bmod m_j$, $s_i(x_j) \equiv 0 \bmod \ell^{L_j - L_i}$

def *lift_simple_root*$(s_i, r_i, u_i, m_i, L_i, \ell, L)$**:**
  while $L_i < L$**:**
    $s_i(x) \leftarrow s_i(\ell \cdot x + r_i)/\ell$
    $L_i \leftarrow L_i + 1$
    $r_i \leftarrow \text{root}_{\mathbb{Z}/\ell\mathbb{Z}}(s_i(x))$ #(by Hensel, a simple root lifts to one root only)
    $u_i \leftarrow u_i + \text{lift}_{\mathbb{N}}(r_i) \cdot m_i$
    $m_i \leftarrow \ell \cdot m_i$
  return (u_i, m_i, L_i)

Algorithm 3.3: Hensel lifting of multiple roots
Input: polynomial $s_i(x) \in \mathbb{Q}[x]$, modulus m_i, residue $u_i \bmod m_i$, multiple root $r_i \bmod \ell$, prime integer ℓ, integer L_i and bound $L > 0$
Output: Residues u_j, integers L_j and moduli m_j s.t. for all $x_j \equiv u_j \bmod m_j$, $s_i(x_j) \equiv 0 \bmod \ell^{L_j - L_i}$

def *lift_multiple_root*$(s_i, r_i, u_i, m_i, L_i, \ell, L)$**:**
  $S \leftarrow [(s_i, r_i, u_i, m_i, L_i)]$ a linked list
  $R \leftarrow \{\}$
  while S *is not empty*:
    $(s_i, r_i, u_i, m_i, L_i) \leftarrow \texttt{pop}(S)$
    $s_i(x) \leftarrow s_i(\ell \cdot x + r_i)/\ell$
    $v_i \leftarrow \text{valuation}_\ell(\text{content}(s_i(x)))$
    $s_i(x) \leftarrow s_i(x)/\ell^{v_i}$
    $L_i \leftarrow L_i + 1 + v_i$
    if $L_i \geq L$**:**
      $R \leftarrow R \cup \{(u_i, m_i, L_i)\}$
    else:
      for $r_i \in \mathbb{Z}/\ell\mathbb{Z}$ *a simple root of* $s_i(x)$ *modulo* ℓ**:**
        $r_i \leftarrow \text{lift}_{\mathbb{N}}(r_i)$
        $(u_j, m_j, L_j) \leftarrow \texttt{lift\_simple\_root}(s_i, r_i, u_i + r_i \cdot m_i, m_i \cdot \ell, L_i, \ell, L)$
        $R \leftarrow R \cup \{(u_j, m_j, L_j)\}$
      for $r_i \in \mathbb{Z}/\ell\mathbb{Z}$ *a multiple root of* $s_i(x)$ *modulo* ℓ**:**
        $r_i \leftarrow \text{lift}_{\mathbb{N}}(r_i)$
        $S \leftarrow \texttt{append}(S, (s_i, r_i, u_i + r_i \cdot m_i, m_i \cdot \ell, L_i))$
  return R

Table 3.3: Security level estimates of MNT curves from [Gui21], and of the GMV curve obtained with the software from [GS21].

curve	k	D	ref	r bits	p bits	p^k bits	DL cost in $\mathbb{F}_{p^k}$
MNT4-298	4	614144978799019	[BCTV14a]	298	298	1192	2^{77}
MNT6-298	6	614144978799019	[BCTV14a]	298	298	1788	2^{87}
MNT4-753	4	241873351932854907	[BCTV14a]	753	753	3012	2^{113}
MNT6-753	6	241873351932854907	[BCTV14a]	753	753	4517	2^{137}
MNT4-992	4	95718723	[Gui21]	992	992	3966	2^{126}
MNT6-992	6	95718723	[Gui21]	992	992	5948	2^{156}
GMV6-183	6	21048712401	[BCG+13]	181	183	1093	2^{71}

in a field $\mathbb{F}_{p^6}$ but new security estimates are not extrapolated. In Table 3.3 we reproduce the key-sizes from [GS21, Gui21].

For pairing-friendly curves with parameters given by polynomials evaluated at some seed, the *Special*-TNFS algorithm applies. It exploits the special form of the prime p, resulting in an asymptotic complexity of $L_{p^k}(1/3, (32/9)^{1/3} \approx 1.526)$ in the most favorable case. It means that compared to prime fields $\mathbb{F}_q$, the total size $k \log p$ should be twice as large to ensure the same level of security: $k \log p = 2 \log q$. For MNT and GMV curves, p is given by a quadratic polynomial and the special variant of TNFS is not better than the generic methods (the degree of $p(x)$ should be at least 3 to make a difference). For the GMV6 given in Table 1.1 (parameters in **bold**), a change of variables $v = -26x - 3$ gives $t(v) = v + 1$, $r(v) = (v^2 - v + 1)/13$ and $p(v) = 4r(v) + t(v) - 1 = (4v^2 + 9v + 4)/13$ with smaller coefficients. We can reasonably assume that the sizes required for GMV6 curves are the same as for MNT6 curves. Such sizes were given in [Gui21, § MNT Curves]. We run the SageMath code of [GS21] and obtain a security estimate of 71 bits in GF(p^6) for the GMV6 curve (Table 3.3). Security levels for BLS12-377, CP6-782 and BW6-761 were provided in [EHG20, § C] (Table 3.4).

Cheon's attack.

Cheon [Che10] showed that given $G, [\tau]G$ and $[\tau^{\mathrm{T}}]G$, with G a point in a subgroup $\mathbb{G}$ of order r with $\mathrm{T} \mid r - 1$, it is possible to recover τ in $2(\lceil\sqrt{(r-1)/\mathrm{T}}\rceil + \lceil\sqrt{\mathrm{T}}\rceil) \times (\mathtt{Exp}_{\mathbb{G}}(r) + \log r \times \mathtt{Comp}_{\mathbb{G}})$ where $\mathtt{Exp}_{\mathbb{G}}(r)$ stands for the cost of one exponentiation in $\mathbb{G}$ by a positive integer less than r and $\mathtt{Comp}_{\mathbb{G}}$ for the cost to determine if two elements are equal in $\mathbb{G}$. According to [Che10, Theorem 2], if $\mathrm{T} \leq r^{1/3}$, then the complexity of the attack is about $O(\sqrt{r/\mathrm{T}})$ exponentiation by using $O(\sqrt{r/\mathrm{T}})$ storage. Sakemi et al. reported an implementation on a 160-bit elliptic curve in [SHI+12].

In SNARKs such as [Gro16] and KZG-based schemes, the `Setup` keys include elements $[\{\tau^i\}_{i=0}^{\mathrm{T}}]G \in \mathbb{G}$ where $\mathrm{T} \in \mathbb{N}^*$ is at least the size of the arithmetic circuit related to the statement to prove, and τ is the secret trapdoor. The property $\mathrm{T} \mid r - 1$ also holds since we need $r - 1$ to be highly *2-adic* (condition (ii)). So, given these auxiliary inputs, an attacker can recover the secret using Cheon's algorithm in time $O(\sqrt{r/\mathrm{T}})$, hence breaking the SNARK soundness. We stress that this attack vector is not inherent to the curve design but to the SNARK design. Given a curve where $r \equiv 1 \bmod 2^L$ used in a SNARK requiring a setup of size $2^{L'}$ where $L' < L$, Cheon's attack runs in $O(\sqrt{r/L'})$ and not $O(\sqrt{r/L})$. To the best of our knowledge, the Filecoin circuit ($L'_{\mathbb{G}_1} = 28, L'_{\mathbb{G}_2} = 27$) is the

Table 3.4: Security level estimates of BLS curves, CP6-782, BW6 and CP8, CP12 curves from [EHG20, EHG22], with seeds u_{377} = `0x8508c00000000001`, u_{379} = `0x9b04000000000001`, $u_{315} = -$`0xbfcfffff`, u_{317} = `0xe19c0001`.

curve	k	D	ref	r bits	p bits	p^k bits	DL cost in $\mathbb{F}_{p^k}$
BLS12-377, u_{377}	12	3	[BCG+20]	253	377	4521	2^{126}
BLS12-379, u_{379}	12	3	[EHG22, Tab. 9]	254	379	4537	2^{126}
BLS24-315, u_{315}	24	3	[EHG22, Tab. 10]	253	315	7543	2^{160}
BLS24-317, u_{317}	24	3	[EHG22, Tab. 10]	255	317	7599	2^{160}
outer curve with a BLS12 curve							
CP6-782, u_{377}	6	339	[BCG+20]	377	782	4691	2^{138}
BW6-761, u_{377}	6	3	[EHG20]	377	761	4566	2^{126}
BW6-764, u_{379}	6	3	[EHG22, Tab. 11]	379	764	4584	2^{126}
outer curves with the BLS24-315 curve							
BW6-633, u_{315}	6	3	[EHG22, Tab. 11]	315	633	3798	2^{124}
BW6-672, u_{315}	6	3	[EHG22, Tab. 11]	315	672	4032	2^{128}
CP8-632, u_{315}	8	4	[EHG22, Tab. 7]	315	632	5056	2^{140}
CP12-630, u_{315}	12	3	[EHG22, Tab. 7]	315	630	7560	2^{166}

Table 3.5: Security level estimates of BN curves and outer curves with the software shipped with [GS21].

curve	k	D	ref	r bits	p bits	p^k bits	DL cost in $\mathbb{F}_{p^k}$
BN-256 $u = 1868033^3$ Pinocchio	12	3	[PHGR13]	256	256	3063	2^{103}
BN-254 $u = 2^{62} - 2^{54} + 2^{44}$ Pantry	12	3	[BFR+13]	256	256	3038	2^{102}
BN-254 $u = -(2^{62} + 2^{55} + 1)$ Geppetto	12	3	[CFH+15]	254	254	3038	2^{102}
BN-254 $u =$ `0x44e992b44a6909f1` Ethereum	12	3	[BCTV14b]	254	254	3044	2^{103}
outer curve with the Geppetto BN curve							
BW6-509	6	3	[CFH+15]	254	509	3051	2^{102}

Table 3.6: Properties of SNARK curves from the literature.

Curve	x	L	$r = \#\mathbb{G}_1$ (bits)	p, $\mathbb{G}_1$ (bits)	$p^{k/d}$, $\mathbb{G}_2$ (bits)	$p \equiv 3 \bmod 4$	security (bits) $\mathbb{G}_1$	$\mathbb{F}^*_{p^k}$
BN-256 [PHGR13]	1868033[3] $\mathrm{HW}_{\text{2-NAF}}(6x+2) = 19$	5	256	256	512	✓	128	103
BN-254 [BFR+13]	$2^{62} - 2^{54} + 2^{44}$ $\mathrm{HW}_{\text{2-NAF}}(6x+2) = 7$	45	254	254	508	×	127	102
GMV6-183 [BCG+13]	`0x8eed757d90615e40000000` $\mathrm{HW}(-26x-2) = 16$	31	181	183	549	NA[2]	90	71
BN-254 [BCTV14b]	`0x44e992b44a6909f1` $\mathrm{HW}_{\text{2-NAF}}(6x+2) = 22$	28	254	254	508	✓	127	103
BLS12-381 [Bow17]	`-0xd201000000010000` $\mathrm{HW}(x) = 6$	32	255	381	762	✓	127	126

biggest application circuit of public interest.

While taking into account this attack at the curve design level might limit the attack vector, this prevents a nice speed up in the pairing computation. As we observed in [EHG22], having a large L s.t. $r(x) \equiv 1 \bmod 2^L$ is often entangled to having a large number of consecutive zeroes in the seed x. This allows mixing efficiently the Karabina [Kar13] and Granger-Scott [GS10] cyclotomic squaring algorithms, hence speeding up significantly the final exponentiation. That is said, Cheon's attack must be taken into account when implementing a SNARK circuit with a given elliptic curve.

3.1.3 Examples

In Table 3.6, we recall the literature curves presented in the state-of-the-art section 2.2 and summarize their SNARK-friendliness properties and security levels.

We see that only the BLS12-381 satisfies conditions (ii), (iii), (iv), (v) and has almost a 128-bit security level. KSS16 and KSS18 families were not investigated in the SNARK literature. We considered the BLS24 family in [EHG22] in the context of 2-chains, hence the proposed BLS24-315 does not satisfy the condition (iii) by definition. In table 3.7, we propose new BN and BLS24 that satisfy all the requirements. We also propose the first KSS16 and KSS18 SNARK curves and compare them to the existing curves. We omit to revisit the GMV6 family as a 128-bit secure curve would be defined over a large field (around 704 bits).

KSS16 and KSS18. The KSS family was not investigated previously in the SNARK context. KSS16 and KSS18 defined respectively over fields of size 328-bit and 348-bit offer 128 bits of security. We suggest in [AEHG22] the KSS16-329 and KSS18-345 that fulfill all conditions except $p \equiv 3 \bmod 4$ (condition (iii) does not apply to KSS18 and is not possible for KSS16).

BLS24. We previously investigated in [EHG22] the BLS family with embedding degree $k = 24$ in the recursive SNARK context. However, we only considered in that paper lifting

[2]For GMV6-183, 3 is the smallest cubic non-residue on $\mathbb{F}_p$.

[3]For KSS18-345, 2 is the smallest cubic non-residue on $\mathbb{F}_p$.

Curve	x	L	$r = \#\mathbb{G}_1$ (bits)	p, $\mathbb{G}_1$ (bits)	$p^{k/d}$, $\mathbb{G}_2$ (bits)	$p \equiv 3 \bmod 4$	security (bits) $\mathbb{G}_1$	$\mathbb{F}^*_{p^k}$
BN383	`0x49e69d16fdc80216226909f1` $\mathrm{HW}_{\text{2-NAF}}(6x+2) = 30$	44	383	383	766	✓	191	123
BLS24-317	`0xd9018000` $\mathrm{HW}_{\text{2-NAF}}(x) = 6$	60	255	317	1268	✓	127	160
KSS16-329	`0x38fab7583` $\mathrm{HW}(x) = 12$	19	255	329	1316	✓	127	140
KSS18-345	`0xc0c44000000` $\mathrm{HW}(x) = 6$	78	254	345	690	NA[3]	127	150

Table 3.7: New SNARK curves from the BN, BLS24, KSS16 and KSS18 families.

of simple roots and proposed the BLS24-315 curve with $2^{22} \mid r-1$. In [AEHG22] we consider multiple roots (cf. 3.3) and propose the BLS24-317 curve with $2^{60} \mid r-1$ that fulfills all SNARK-friendliness conditions. This curve is particularly suitable for KZG-based SNARKs compared to previous known curves. As a reference example, compared to the widely used BLS12-381, it takes 14% less time to generate a PLONK proof of a circuit of 40000 constraints (implemented in `gnark` [BPH+22a]). The setup generation is also 23% faster but the verification is 30% slower, although this can be likely amortized with a batch verification. The verification overhead in BLS24 is due to the cost of $\mathbb{F}_{p^{24}}$ arithmetic in the pairing computation. However, this can be somewhat reduced using a 2^n-tuple-and-add Miller loop following [CBGW10], which we have not implemented in `gnark-crypto` yet at the time of writing.

3.2 Efficient arithmetic

In this section, we focus on two important cryptographic operations: Hashing to an elliptic curve and subgroup membership. These optimizations work for a wide range of elliptic curves including in particular SNARK-friendly curves. Hash from a (random) string to a point on the elliptic curve is an important cryptographic operation. It has two steps: first mapping a string to a point $P(x, y)$ on the curve, then multiplying the point by the cofactor so that it falls into the cryptographic subgroup. For the first step, there is the efficient Elligator function for curves with j-invariant not 0 nor 1728 and having a point of order 4. For other curves including BLS curves of j-invariant 0, Wahby and Boneh propose an efficient map in [WB19]. Because the BLS12-381 curve is not of prime order, the point is multiplied by the cofactor c_1 to ensure the hash function to map into the cryptographic subgroup of 255-bit prime order. Wahby and Boneh wrote in [WB19] that it is sufficient to multiply by $(x-1)$, instead of the cofactor $(x-1)^2/3$. They observed that for any prime factor ℓ of $(x-1)$, the BLS12-381 curve has no point of order ℓ^2. We prove this trick and show that it works for a wide range of elliptic curves.

Another important operation is to test whether a given point belongs to the right subgroup of order r, i.e. $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_T$. This is a crucial operation to avoid small subgroups attacks. This test can be done much faster if an efficient endomorphism is available, which is usually the case for pairing-friendly curves.

3.2.1 Faster co-factor clearing

Let $\mathrm{End}_{\mathbb{F}_p}(E)$ denote the ring of $\mathbb{F}_p$-endomorphisms of E, let $\mathcal{O}$ denotes a complex quadratic order of the ring of integers of a complex quadratic number field, and $\mathcal{O}(\Delta)$ denotes the complex quadratic order of discriminant Δ.

Theorem 3.1 ([Sch87, Proposition 3.7]). *Let E be an elliptic curve over $\mathbb{F}_p$ and $n \in \mathbb{Z}_{\geq 1}$ with $p \nmid n$. Let π_p denote the Frobenius endomorphism of E of trace t. Then,*

$$E[n] \subset E(\mathbb{F}_p) \iff \begin{cases} n^2 \mid \#E(\mathbb{F}_p), \\ n \mid p-1 \text{ and} \\ \pi_p \in \mathbb{Z} \text{ or } \mathcal{O}\big(\frac{t^2-4p}{n^2}\big) \subset \mathrm{End}_{\mathbb{F}_p}(E). \end{cases}$$

In [EHGP22], we apply this theorem to the polynomial families of the taxonomy paper of Freeman, Scott and Teske [FST10]. The families are designed for specific discriminants $D = 1$ for constructions 6.2, 6.3 and 6.4, $D = 3$ for construction 6.6 and some of the KSS families, $D = 2$ for construction 6.7. First we identify a common cofactor within the family which has a square factor, then we compute its gcd with $p(x) - 1$ and $y(x)$. We summarize our results in the following tables and provide a SageMath verification script at `https://gitlab.inria.fr/zk-curves/cofactor`.

Construction 6.6

The family of pairing-friendly BLS curves appeared in [BLS03]. A BLS curve can have an embedding degree k multiple of 3 but not 18. Common examples are $k = 9, 12, 15, 24, 27, 48$. A generalization was given in [FST10] and named Construction 6.6. Let k be a positive integer with $k \leq 1000$ and $18 \nmid k$. Construction 6.6 is given in Table 3.8. Then (t, r, p) parameterizes a complete family of pairing-friendly curves with embedding degree k and discriminant 3. Next, in Table 3.9, we compute the cofactor polynomial $c_1(x)$ for Construction 6.6 family. We recall that $y(x)$ satisfies the Complex Multiplication equation $4p(x) = t(x)^2 + Dy(x)^2$. To prove the results of Table 3.9, we will need some basic polynomial results that we prove in Lemmas 3.1, 3.2, 3.3, and 3.4.

Lemma 3.1. *Over the field of rationals $\mathbb{Q}$, $\Phi_d(x)$ denotes the d-th cyclotomic polynomial, and for all the distinct divisors d of n including 1 and n,*

$$x^n - 1 = \prod_{d \mid n} \Phi_d(x) \ . \tag{3.1}$$

Lemma 3.2. *For any odd $k \geq 1$ not multiple of 3 ($k \equiv 1, 5 \bmod 6$), we have*

$$x^2 - x + 1 \mid x^{2k} - x^k + 1 \ . \tag{3.2}$$

Proof (of Lemma 3.2). By Lemma 3.1, $x^{6k} - 1$ is a multiple of $\Phi_1 = x - 1$, $\Phi_2 = x + 1$, $\Phi_3 = x^2 + x + 1$ and $\Phi_6 = x^2 - x + 1$. Since

$$x^{6k} - 1 = (x^{3k} - 1)(x^{3k} + 1) = (x^k - 1)(x^{2k} + x^k + 1)(x^k + 1)(x^{2k} - x^k + 1)$$

and $\Phi_1\Phi_3 \mid x^{3k} - 1$ but $\Phi_6 \nmid x^{3k} - 1$ because k is odd, nor $x^k + 1$ because k is not multiple of 3, then $\Phi_6 = x^2 - x + 1$ should divide the other term $x^{2k} - x^k + 1$. □

Table 3.8: Construction 6.6 from [FST10, §6], formulas for $k = 9, 15 \bmod 18$ from ePrint.

k	$r(x)$	$t(x)$	$y(x)$	$p(x)$	$x \bmod 3$
$1 \bmod 6$	$\Phi_{6k}(x)$	$-x^{k+1} + x + 1$	$(-x^{k+1} + 2x^k - x - 1)/3$	$(x+1)^2(x^{2k} - x^k + 1)/3 - x^{2k+1}$	2
$2 \bmod 6$	$\Phi_{3k}(x)$	$x^{k/2+1} - x + 1$	$(x^{k/2+1} + 2x^{k/2} + x - 1)/3$	$(x-1)^2(x^k - x^{k/2} + 1)/3 + x^{k+1}$	1
$3 \bmod 18$	$\Phi_{2k}(x)$	$x^{k/3+1} + 1$	$(-x^{k/3+1} + 2x^{k/3} + 2x - 1)/3$	$(x^2 - x + 1)^2(x^{2k/3} - x^{k/3} + 1)/3 + x^{k/3+1}$	2
$9, 15 \bmod 18$	$\Phi_{2k}(x)$	$-x^{k/3+1} + x + 1$	$(-x^{k/3+1} + 2x^{k/3} - x - 1)/3$	$(x+1)^2(x^{2k/3} - x^{k/3} + 1)/3 - x^{2k/3+1}$	2
$4 \bmod 6$	$\Phi_{3k}(x)$	$x^3 + 1$	$(x^3 - 1)(2x^{k/2} - 1)/3$	$(x^3 - 1)^2(x^k - x^{k/2} + 1)/3 + x^3$	1
$5 \bmod 6$	$\Phi_{6k}(x)$	$x^{k+1} + 1$	$(-x^{k+1} + 2x^k + 2x - 1)/3$	$(x^2 - x + 1)(x^{2k} - x^k + 1)/3 + x^{k+1}$	2
$0 \bmod 6$	$\Phi_k(x)$	$x + 1$	$(x - 1)(2x^{k/6} - 1)/3)$	$(x-1)^2(x^{k/3} - x^{k/6} + 1)/3 + x$	1

Lemma 3.3. *For any odd $k \geq 1$ such that ($k \equiv 1 \bmod 6$), we have*

$$x^2 - x + 1 \mid x^{k+1} - x + 1 \quad \textit{and} \quad x^2 - x + 1 \mid x^{k+1} - 2x^k + x + 1 \ . \tag{3.3}$$

Proof (of Lemma 3.3). Let $\omega, \overline{\omega} \in \mathbb{C}$ be the two primitive 6-th roots of unity that are the two roots of $x^2 - x + 1$. Since $k \equiv 1 \bmod 6$ and $\omega^6 = \overline{\omega}^6 = 1$, then $\omega^k = \omega, \overline{\omega}^k = \overline{\omega}, \omega^{k+1} = \omega^2$ and $\overline{\omega}^{k+1} = \overline{\omega}^2$. Then $\omega^{k+1} - \omega + 1 = \omega^2 - \omega + 1 = 0$ and $\overline{\omega}^{k+1} - \overline{\omega} + 1 = \overline{\omega}^2 - \overline{\omega} + 1 = 0$. Hence $\omega, \overline{\omega}$ are roots of $x^{k+1} - x + 1$ and $x^2 - x + 1$ divides $x^{k+1} - x + 1$. Similarly, $\omega^{k+1} - 2\omega^k + \omega + 1 = \omega^2 - 2\omega + \omega + 1 = 0$ and the same holds for $\overline{\omega}$. We conclude that $x^2 - x + 1$ divides $x^{k+1} - 2x^k + x + 1$. □

Lemma 3.4. *For any odd $k \geq 1$ such that ($k \equiv 5 \bmod 6$), we have*

$$x^2 - x + 1 \mid x^{k+1} - 2x^k - 2x + 1 \ . \tag{3.4}$$

Proof (of Lemma 3.4). Let $\omega, \overline{\omega} \in \mathbb{C}$ be the two primitive 6-th roots of unity that are the two roots of $x^2 - x + 1$. Similarly as in the proof of Lemma 3.3, since $k \equiv 5 \bmod 6$ and $\omega^3 = -1$, $\omega^6 = 1$, then $\omega^{k+1} = 1$, $\omega^k = \omega^5 = -\omega^2$. Then $\omega^{k+1} - 2\omega^k - 2\omega + 1 =$

Table 3.9: Cofactors of Construction 6.6 families

k	$p(x) + 1 - t(x)$	$c_0(x)$	$\gcd(c_0(x), p(x) - 1)$	$\gcd(c_0(x), y(x))$
$1 \bmod 6$	$(x^{2k} - x^k + 1)(x^2 - x + 1)/3$	$(x^2 - x + 1)^2/3$	$x^2 - x + 1$	$(x^2 - x + 1)/3$
$2 \bmod 6$	$(x^k - x^{k/2} + 1)(x^2 + x + 1)/3$	$(x^2 + x + 1)/3$	1	1
$3 \bmod 18$	$(x^{2k/3} - x^{k/3} + 1)(x^2 - x + 1)^2/3$	$(x^2 - x + 1)^2/3$	1	1
$9 \bmod 18$	$(x^{2k/3} - x^{k/3} + 1)(x^2 - x + 1)/3$	$(x^2 - x + 1)/3$	1	1
$15 \bmod 18$	$(x^{2k/3} - x^{k/3} + 1)(x^2 - x + 1)/3$	$(x^2 - x + 1)^2/3$	1	1
$4 \bmod 6$	$(x^k - x^{k/2} + 1)(x^3 - 1)^2/3$	$(x^3 - 1)^2/3$	$x^3 - 1$	$(x^3 - 1)/3$
$5 \bmod 6$	$(x^{2k} - x^k + 1)(x^2 - x + 1)/3$	$(x^2 - x + 1)^2/3$	$x^2 - x + 1$	$(x^2 - x + 1)/3$
$0 \bmod 6$	$(x^{k/3} - x^{k/6} + 1)(x - 1)^2/3$	$(x - 1)^2/3$	$x - 1$	$(x - 1)/3$

$1 - 2(-\omega^2) - 2\omega + 1 = 2\omega^2 - 2\omega + 2 = 0$. The same holds for $\overline{\omega}$, and we conclude that $x^2 - x + 1$ divides $x^{k+1} - 2x^k - 2x + 1$. □

Proof (of Table 3.9). For $k = 1 \bmod 6$, one computes

$$\begin{aligned} p(x) + 1 - t(x) =& (x+1)^2(x^{2k} - x^k + 1)/3 - x^{2k+1} + 1 - (-x^{k+1} + x + 1) \\ =& (x+1)^2(x^{2k} - x^k + 1)/3 - x(x^{2k} - x^k + 1) \\ =& (x^{2k} - x^k + 1)(x^2 - x + 1)/3 \ . \end{aligned}$$

By Lemma 3.2, (x^2-x+1) divides $x^{2k}-x^k+1$ since $k \equiv 1 \bmod 6$. Note that for $x \equiv 2 \bmod 3$, $x^2 - x + 1 \equiv 0 \bmod 3$. Hence the cofactor is a multiple of $c_0(x) = (x^2 - x + 1)^2/3$. Next, one computes

$$\begin{aligned} p(x) - 1 =& \underbrace{(x+1)^2}_{=(x^2-x+1)+3x} (x^{2k} - x^k + 1)/3 - x^{2k+1} - 1 \\ =& (x^2 - x + 1)(x^{2k} - x^k + 1)/3 + x(x^{2k} - x^k + 1) - x^{2k+1} - 1 \\ =& (x^2 - x + 1)(x^{2k} - x^k + 1)/3 - (x^{k+1} - x + 1) \end{aligned}$$

and by Lemma 3.3, $x^2 - x + 1$ divides $x^{k+1} - x + 1$. We computed the derivative of $p(x) - 1$ and checked that none of $\omega, \overline{\omega}$ is a zero of the derivative. Finally, $x^2 - x + 1$ divides $p(x) - 1$ with multiplicity one. To conclude, Lemma 3.3 ensures that $(x^2 - x + 1)$ divides $y(x)$, and we checked that the derivative of $y(x)$ does not vanish at a primitive sixth root of unity, hence $x^2 - x + 1$ divides $y(x)$ with multiplicity one.

For $k = 2 \bmod 6$, one computes

$$\begin{aligned} p(x) + 1 - t(x) =& (x-1)^2(x^k - x^{k/2} + 1)/3 + x^{k+1} + 1 - (x^{k/2+1} - x + 1) \\ =& (x^2 - 2x + 1)(x^k - x^{k/2} + 1)/3 + x(x^k - (x^{k/2} + 1) \\ =& (x^k - x^{k/2} + 1)(x^2 + x + 1)/3 \end{aligned}$$

Note that k is even. Lemma 3.2 will apply for $k' = k/2$ to be odd, that is $k \equiv 2 \bmod 12$. Nevertheless the cofactor $c_0(x)$ will not be a square. We checked that none of the primitive cubic and sextic roots of unity are roots of $p(x) - 1$ nor $y(x)$, hence the gcd of $c_0(x)$ and $p(x) - 1$, resp. $y(x)$, is 1.

For $k = 3 \bmod 18$, it is straightforward to get $p(x) + 1 - t(x) = (x^2 - x + 1)^2(x^{2k/3} - x^{k/3} + 1)/3$, the cofactor $c_0(x) = (x^2 - x + 1)^2/3$ is a square as for $k = 1 \bmod 6$. For $k = 9, 15 \bmod 18$, we compute

$$\begin{aligned} p(x) + 1 - t(x) =& (x+1)^2(x^{2k/3} - x^{k/3} + 1)/3 - x^{2k/3+1} + 1 - (-x^{k/3+1} + x + 1) \\ =& (x^2 + 2x + 1)(x^{2k/3} - x^{k/3} + 1)/3 - x(x^{2k/3} - x^{k/3} + 1) \\ =& (x^2 - x + 1)(x^{2k/3} - x^{k/3} + 1)/3 \end{aligned}$$

For $k = 9 \bmod 18$, $k/3$ is a multiple of 3 and $x^2 - x + 1$ does not divide $(x^{2k/3} - x^{k/3} + 1)$, while for $k = 15 \bmod 18$, $k/3$ is co-prime to 6, and $(x^{2k/3} - x^{k/3} + 1)$ is a multiple of $(x^2 - x + 1)$ by Lemma 3.2. For $k \equiv 3, 9, 15 \bmod 18$, we checked that neither $p(x) - 1$ nor $y(x)$ have a common factor with $c_0(x)$, and no faster co-factor clearing is available.

For $k \equiv 4, 0 \bmod 6$, the calculus is similar to the case $k \equiv 1 \bmod 6$, and for $k \equiv 5 \bmod 6$, we use Lemma 3.4 to conclude about $y(x)$. □

For the cases $k \equiv 2 \mod 6$ and $k \equiv 9 \mod 18$, $c_1(x)$ has no square factor and thus the cofactor clearing is already optimised. For $k \equiv 3, 15 \bmod 18$, the cofactor is a square but Theorem 3.1 does not apply. For all remaining cases, $c_1(x) = n(x)^2/3$ for some polynomial factor $n(x)/3$ that satisfies Theorem 3.1. Hence, it is sufficient to multiply by $n(x)$ to clear the cofactor on Construction 6.6 curves. We summarize our results in Theorem 3.2.

Theorem 3.2.
- *For $k \equiv 1, 5 \bmod 6$, the curve cofactor has a factor $c_0(x) = (x^2 - x + 1)^2/3$, whose structure is $\mathbb{Z}/(x^2 - x + 1)/3\mathbb{Z} \times \mathbb{Z}/(x^2 - x + 1)\mathbb{Z}$, and it is enough to multiply by $n(x)$ to clear the co-factor $c_0(x)$.*
- *For $k \equiv 4 \bmod 6$, the curve cofactor has a factor $c_0(x) = (x^3-1)^2/3$, whose structure is $\mathbb{Z}/(x^3 - 1)/3\mathbb{Z} \times \mathbb{Z}/(x^3 - 1)\mathbb{Z}$, and it is enough to multiply by $n(x)$ to clear the co-factor $c_0(x)$.*
- *For $k \equiv 0 \bmod 6$, the curve cofactor has a factor $c_0(x) = (x-1)^2/3$, whose structure is $\mathbb{Z}/(x - 1)/3\mathbb{Z} \times \mathbb{Z}/(x - 1)\mathbb{Z}$, and it is enough to multiply by $n(x)$ to clear the co-factor $c_0(x)$.*

Proof (of Th. 3.2). From Table 3.9, $k = 1, 5 \bmod 6$ has $n(x) = (x^2-x+1)/3$, $k = 4 \bmod 6$ has $n(x) = (x^3 - 1)/3$, $k = 0 \bmod 6$ has $n(x) = (x - 1)/3$ where $n(x)$ satisfies the conditions of Th. 3.1. The n-torsion is $\mathbb{F}_p$-rational, that is $E[n] \subset E(\mathbb{F}_p)$ and has structure $\mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ over $\mathbb{F}_p$. Taking into account the co-factor 3, the structure of the subgroup of order $c_0(x) = 3n^2(x)$ is $\mathbb{Z}/3n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$ and multiplying by $3n(x)$ clears the cofactor. □

Example 3.1. *In [CDS20], Clarisse, Duquesne and Sanders introduced two new pairing-friendly curves with optimal $\mathbb{G}_1$, the curves BW13-P310 with seed $u = -$`0x8b0` and BW19-P286 with seed $v = -$`0x91`. They fall in Construction 6.6 with $k = 1 \bmod 6$. Our faster co-factor clearing method applies.*

For BW13-P310, the prime subgroup order is $r = \Phi_{6\cdot 13}(u) = (u^{26} - u^{13} + 1)/(u^2 - u + 1)$. The cofactor is $(u^2 - u + 1)^2/3$, where $(u^2 - u + 1)$ divides $p(u) - 1$ and $(u^2 - u + 1)/3$ divides $y(u)$. It is enough to multiply by $(u^2 - u + 1)$ to clear the cofactor.

For BW19-P286, the prime subgroup order is $r = \Phi_{6\cdot 19}(v) = (v^{38} - v^{19} + 1)/(v^2 - v + 1)$. The cofactor is $(v^2 - v + 1)^2/3$, where $(v^2 - v + 1)$ divides $p(v) - 1$ and $(v^2 - v + 1)/3$ divides $y(v)$. It is enough to multiply by $(v^2 - v + 1)$ to clear the cofactor.

Subgroup Security, Distortion Map. Theorem 3.1 applied to BLS curves tells us that the curve endomorphism $\phi\colon E \to E$, $(x, y) \mapsto (\omega x, y)$ with $\omega \in \mathbb{F}_p$ a primitive third root of unity ($\omega^2 + \omega + 1 = 0 \bmod p$) acts as a *distortion map* on $E[n] \simeq \mathbb{Z}/n\mathbb{Z} \oplus \mathbb{Z}/n\mathbb{Z}$. With a Weil pairing e_W, one can embed a discrete logarithm on $E(\mathbb{F}_p)[n]$ into $\mathbb{F}_p^*$, where sub-exponential DL computation takes place, although the much larger size of p compared to n seems prohibitive. For $G, P \in E[n]$ in the same subgroup of order n, $\log_G(P) = \log_{e_W(G,\phi(G))} e_W(P, \phi(G))$.

The definition of subgroup security in [BCM+15] is the following.

Definition 3.1 (Subgroup Security, [BCM+15, Definition 1])**.** *Let $p(u), t(u), r(u) \in \mathbb{Q}[u]$ parameterize a family of ordinary pairing-friendly elliptic curves, and for any particular $u_0 \in \mathbb{Z}$ such that $p = p(u_0)$ and $r = r(u_0)$ are prime, let E be the resulting pairing-friendly elliptic curve over $\mathbb{F}_p$ of order divisible by r. Let $h_1 = \#E(\mathbb{F}_p)/r$, $h_2 = \#E'(\mathbb{F}_{p^{k/d}})/r$ and $h_T = \Phi_k(p)/r$. We say that E is subgroup-secure if all $\mathbb{Q}[u]$-irreducible factors of $h_1(u), h_2(u)$ and $h_T(u)$ that can represent primes and that have degree at least that of $r(u)$, contain no prime factors smaller than $r(u_0) \in \mathbb{Z}$ when evaluated at $u = u_0$.*

If $c_0 = (x-1)/3$ is prime, since the structure of the subgroup of order c_0^2 is $\mathbb{Z}/c_0\mathbb{Z} \oplus \mathbb{Z}/c_0\mathbb{Z}$, and the subgroup is fully defined over the prime field $\mathbb{F}_p$, one can find a basis $\langle P_1, P_2 \rangle$ so that P_1, P_2 are of order c_0 and linearly independent. Moreover there exists a distortion map ψ from the subgroup $\langle P_1 \rangle$ to $\langle P_2 \rangle$. The distortion map ψ is given by $(x, y) \mapsto (\omega x, y)$ where $\omega \in \mathbb{F}_p$ is such that $\omega^2 + \omega + 1 = 0$. (See [Cha06] on distortion maps on embedding degree 1 curves). Because of this distortion map, one can transfer as in the MOV attack a discrete logarithm computation in the subgroup of order $(x-1)/3$ of $E(\mathbb{F}_p)$ to a discrete logarithm computation in the subgroup of order $(x-1)/3$ of $\mathbb{F}_p$ (note that this is the base field $\mathbb{F}_p$, not the extension field $\mathbb{F}_{p^k}$), where sub-exponential DL computation takes place. The DL computation in $\mathbb{F}_p$ has complexity $\exp\Big((1+o(1))\sqrt{\ln p \ln\ln p}\Big)$ with the quadratic sieve, and $\exp\Big((1.923+o(1))\sqrt[3]{\ln p (\ln\ln p)^2}\Big)$ with the number field sieve. Because the complexity is in p not c_0, the computation will be slower, nevertheless it exists. In practice, if an implementation of a generic DL computation algorithm like Pollard-ρ is faster in $\mathbb{F}_p$ than on $E(\mathbb{F}_p)$ for the subgroup of order $(x-1)/3$, it is possible to transfer the computation from the curve to the finite field thanks to the distortion map and a Weil pairing.

Constructions 6.2, 6.3, 6.4, and 6.5 with $D = 1$

The constructions with numbers 6.2 to 6.5 have discriminant $D = 1$, we report the polynomial forms of the parameters in Table 3.10. The cofactor $c_1(x)$ in $p(x) + 1 - t(x) = r(x)c_1(x)$ has always a factor $c_0(x)$ that we report in Table 3.11, with special cases for $k = 2$ and $k = 4$. For $p(x)$ to be an integer, $x \equiv 1 \bmod 2$ is required, except for 6.5 where x is required to be even.

Table 3.10: Constructions 6.2, 6.3, 6.4, and 6.5 from [FST10, §6]

	k	$r(x)$	$t(x)$	$y(x)$	$p(x)$
6.2	1 mod 2	$\Phi_{4k}(x)$	$-x^2+1$	$x^k(x^2+1)$	$(x^{2k+4}+2x^{2k+2}+x^{2k}+x^4-2x^2+1)/4$
6.3	2 mod 4	$\Phi_{2k}(x)$	x^2+1	$x^{k/2}(x^2-1)$	$(x^{k+4}-2x^{k+2}+x^k+x^4+2x^2+1)/4$
6.4	4 mod 8	$\Phi_k(x)$	$x+1$	$x^{k/4}(x-1)$	$(x^{k/2+2}-2x^{k/2+1}+x^{k/2}+x^2+2x+1)/4$
6.5	$k=10$	$\Phi_{20}(x)$	$-x^6+x^4-x^2+2$	$x^3(x^2-1)$	$(x^{12}-x^{10}+x^8-5x^6+5x^4-4x^2+4)/4$

Table 3.11: Cofactors of Constructions 6.2, 6.3, 6.4, and 6.5. Note that $x \equiv 1 \bmod 2$ except for 6.5 where $x \equiv 0 \bmod 2$.

	k	$c_0(x)$	$\gcd(c_0(x), p(x)-1)$	$\gcd(c_0(x), y(x))$
6.2	1 mod 2	$(x^2+1)^3/4$	x^2+1	x^2+1
6.3	$k=2$	$(x^2-1)^2/2$	x^2-1	x^2-1
6.3	2 mod 4, $k>2$	$(x^2-1)^2(x^2+1)/4$	x^2-1	x^2-1
6.4	$k=4$	$(x-1)^2/2$	$x-1$	$x-1$
6.4	4 mod 8, $k>4$	$(x-1)^2(x^2+1)/4$	$x-1$	$x-1$
6.5	$k=10$	$x^4/4$	x^2	x^3

Lemma 3.5. *For any odd $k \geq 1$ we have*

$$x^2 + 1 \mid x^{2k} + 1 \ . \tag{3.5}$$

Explicitly,

$$x^{2k} + 1 = (x^2+1)(1 - x^2 + x^4 - \ldots + \ldots - x^{2k-4} + x^{2k-2}) \ . \tag{3.6}$$

Proof. By Lemma 3.1, $x^{4k}-1$ is a multiple of $\Phi_1 = x-1$, $\Phi_2 = x+1$ and $\Phi_4 = x^2+1$. Since $x^{4k}-1 = (x^{2k}-1)(x^{2k}+1)$ and $\Phi_1\Phi_2 \mid x^{2k}-1$ but $\Phi_4 \nmid x^{2k}-1$ because k is odd, then $\Phi_4 = x^2+1$ should divide the other term $x^{2k}+1$. □

Proof (of Table 3.11). All families of constructions 6.2 to 6.5 have j-invariant 1728, an a point of order 2 (their order is even).

In Construction 6.2 one has k odd. One gets $p(x)+1-t(x) = (x^2+1)^2(x^{2k}+1)/4$, and by Lemma 3.5, x^2+1 is a factor of $x^{2k}+1$, hence $c_0(x) = (x^2+1)^3/4$ which is even, divides $p(x)+1-t(x)$. The factorization of $p(x)-1$ is

$$\begin{aligned}
p(x)-1 =& (x^{2k}(x^2+1)^2 + (x^2-1)^2)/4 - 1\\
=& ((x^4+2x^2+1)x^{2k} + (x^4-2x^2+1) - 4)/4\\
=& ((x^4-1)x^{2k} + (2x^2+2)x^{2k} + (x^4-1) - 2x^2 - 2)/4\\
=& ((x^4-1)(x^{2k}+1) + 2(x^2+1)(x^{2k}-1))/4\\
=& (x^4-1)(x^{2k}+1+2a(x))/4 \text{ where}\\
a(x) =& (x^{2k}-1)/(x^2-1) = 1 + x^2 + x^4 + \ldots + x^{2k-2} = \sum_{i=0}^{k-1} x^{2i}
\end{aligned}$$

and by Lemma 3.1, $x^{2k}-1$ is a multiple of $x^2-1 = \Phi_1\Phi_2$, and $(x^4-1)/2$ divides $p(x)-1$. More precisely, because x is odd, $4 \mid p(x)-1$, and

$$p(x)-1 = 2\underbrace{(x^2+1)}_{\text{even}}\underbrace{(x^2-1)/4}_{\in\mathbb{Z}}\underbrace{(x^{2k}+1+2a(x))/2}_{\in\mathbb{Z}} .$$

As a consequence, x^2+1 divides $p(x)-1$. Finally, $y(x) = x^k(x^2+1)$ is a multiple of x^2+1.

We isolate the case $k=2$ in Construction 6.3, with parameters $r(x) = \Phi_4(x) = x^2+1$ (even), $t(x) = x^2+1$, $y(x) = x(x^2-1)$, $p(x) = (x^6-x^4+3x^2+1)/4$, $p(x)+1-t(x) = (x^2+1)(x^2-1)^2/4$. We set $r(x) = (x^2+1)/2$ and $c_1(x) = (x^2-1)^2/2$, $p(x)-1 = (x^2-1)(x^4+3)/4$ where $(x^4+3)/4$ is an integer. For larger $k = 2 \bmod 4$, one has

$$\begin{aligned}
p(x)+1-t(x) =& (x^{k+4} - 2x^{k+2} + x^k + x^4 + 2x^2 + 1)/4 + 1 - (x^2+1)\\
=& (x^k(x^2-1)^2 + (x^2+1)^2 - 4x^2)/4\\
=& (x^k+1)(x^2-1)^2/4
\end{aligned}$$

and since k is even, by Lemma 3.5, x^2+1 divides x^k+1, hence $c_0(x) = (x^2+1)(x^2-1)^2/4$ divides the curve order. We compute $p(x)-1$ and factor it:

$$\begin{aligned}
p(x)-1 =& (x^k(x^2-1)^2 + (x^2+1)^2)/4 - 1\\
=& (x^k(x^2-1)^2 + (x^2-1)^2 + 4x^2 - 4)/4\\
=& (x^2-1)(x^k\underbrace{(x^2-1)}_{\text{mult. of 4}} + \underbrace{x^2-1}_{\text{mult. of 4}} + 4)/4
\end{aligned}$$

which proves that x^2-1 divides $p(x)-1$. Because $y(x) = x^{k/2}(x^2-1)$, it is obvious that x^2-1 divides $y(x)$.

With Construction 6.4, $k = 4 \bmod 8$. First $k=4$ is a special case where the curve order is $p(x)+1-t(x) = (x-1)^2(x^2+1)/4$, the cofactor is $c_0(x) = (x-1)^2/2$, $r(x) = (x^2+1)/2$,

$p(x)-1=(x^2-1)(x^2-2x+3)/4$ factors as $p(x)-1=(x-1)(x+1)/2(x^2-2x+3)/2$, and $y(x)=x(x-1)$.

For larger k, we compute, with $p(x)=(x^{k/2}(x-1)^2+(x+1)^2)/4$,

$$\begin{aligned} p(x)+1-t(x) =&(x^{k/2}(x-1)^2+(x+1)^2)/4+1-(x+1)\\ =&(x^{k/2}(x-1)^2+x^2+2x+1-4x)/4\\ =&(x^{k/2}(x-1)^2+(x-1)^2)/4\\ =&(x-1)^2(x^{k/2}+1)/4 \end{aligned}$$

and because $k\equiv 4 \bmod 8$, $k/2$ is even and by Lemma 3.5, x^2+1 divides $x^{k/2}+1$, hence $c_0(x)=(x-1)^2(x^2+1)/4$ divides the curve order. Now we compute $p(x)-1$ and obtain the factorisation

$$\begin{aligned} p(x)-1 =&(x^{k/2}(x-1)^2+(x+1)^2)/4-1\\ =&(x^{k/2}(x-1)^2+x^2-2x+1+4x-4)/4\\ =&(x^{k/2}(x-1)^2+(x-1)^2+4(x-1))/4\\ =&(x-1)(x^{k/2}(x-1)+(x-1)+4)/4\\ =&(x-1)(\underbrace{(x^{k/2}+1)(x-1)}_{\text{mult. of 4}}+4)/4 \end{aligned}$$

hence $x-1$ divides $p(x)-1$. Finally $y(x)=x^{k/4}(x-1)$ and $(x-1)$ divides $y(x)$.

For construction 6.5, x is even this time, the curve order is $p(x)+1-t(x)=x^4/4(x^8-x^6+x^4-x^2+1)$, $y(x)=x^3(x^2-1)$, $p(x)-1=x^2(x^{10}-x^8+x^6-5x^4+5x^2-4)/4$ were the factor $(x^{10}-x^8+x^6-5x^4+5x^2-4)/4$ is an integer whenever x is even. □

From Table 3.11 and Theorem 3.1, we obtain Theorem 3.3.

Theorem 3.3.

- *For construction 6.2, the curve cofactor has a factor $c_0(x)=(x^2+1)^3/4$, whose structure is $\mathbb{Z}/(x^2+1)/2\mathbb{Z}\times\mathbb{Z}/(x^2+1)^2/2\mathbb{Z}$, and it is enough to multiply by $n(x)$ to clear the co-factor $c_0(x)$.*
- *For construction 6.3, the curve cofactor has a factor $c_0(x)=(x^2-1)^2(x^2+1)/4$, whose structure is $\mathbb{Z}/(x^2-1)/2\mathbb{Z}\times\mathbb{Z}/((x^2-1)(x^2+1)/2\mathbb{Z}$, and it is enough to multiply by $n(x)$ to clear the co-factor $c_0(x)$.*
- *For construction 6.4, the curve cofactor has a factor $c_0(x)=(x-1)^2(x^2+1)/4$, whose structure is $\mathbb{Z}/(x-1)/2\mathbb{Z}\times\mathbb{Z}/(x-1)(x^2+1)/2\mathbb{Z}$, and it is enough to multiply by $n(x)$ to clear the co-factor $c_0(x)$.*
- *For construction 6.5, the curve order has cofactor $c_0(x)=x^4/4$, whose structure is $\mathbb{Z}/x^2/2\mathbb{Z}\times\mathbb{Z}/x^2/2\mathbb{Z}$, and it is enough to multiply by $n(x)$ to clear the cofactor.*

Construction 6.7 with $D=2$

Construction 6.7 in [FST10] has discriminant $D=2$. We report the polynomial forms of the parameters in Table 3.12. The cofactor $c_1(x)$ in $p(x)+1-t(x)=r(x)c_1(x)$ has always a factor $c_0(x)$ that we report in Table 3.13. For $p(x)$ to be an integer, $x\equiv 1 \bmod 2$ is required, and $x\equiv 1 \bmod 4$ for $k\equiv 0 \bmod 24$.

Table 3.12: Construction 6.7 from [FST10, §6].

6.7, $k = 0 \bmod 3$, $\ell = \mathrm{lcm}(8, k)$	
$r(x) =$	$\Phi_\ell(x)$
$t(x) =$	$x^{\ell/k} + 1$
$y(x) =$	$(1 - x^{\ell/k})(x^{5\ell/24} + x^{\ell/8} - x^{\ell/24})/2$
$p(x) =$	$(2(x^{\ell/k} + 1)^2 + (1 - x^{\ell/k})^2(x^{5\ell/24} + x^{\ell/8} - x^{\ell/24})^2)/8$

Table 3.13: Cofactor of Construction 6.7. Note that $x \equiv 1 \bmod 2$, except for $k \equiv 0 \bmod 24$, where $x \equiv 1 \bmod 4$.

	k	$c_0(x)$	$\gcd(c_0(x), p(x) - 1)$	$\gcd(c_0(x), y(x))$
6.7	0 mod 3	$(x^{\ell/k} - 1)^2/8$	$(x^{\ell/k} - 1)/2$	$(x^{\ell/k} - 1)/2$

Proof (of Table 3.13). We compute

$$\begin{aligned} p(x) + 1 - t(x) =& (2(x^{\ell/k} + 1)^2 + (1 - x^{\ell/k})^2(x^{5\ell/24} + x^{\ell/8} - x^{\ell/24})^2)/8 + 1 - (x^{\ell/k} + 1) \\ =& (2(x^{\ell/k} + 1)^2 - 8x^{\ell/k} + (1 - x^{\ell/k})^2(x^{\ell/24}(x^{4\ell/24} + x^{2\ell/24} - 1))^2)/8 \\ =& (2(x^{\ell/k} - 1)^2 + (x^{\ell/k} - 1)^2 x^{\ell/12}(x^{\ell/6} + x^{\ell/12} - 1)^2)/8 \\ =& (x^{\ell/k} - 1)^2(x^{\ell/12}(x^{\ell/6} + x^{\ell/12} - 1)^2 + 2)/8 \end{aligned}$$

and for $p(x) - 1$ we obtain

$$\begin{aligned} p(x) - 1 =& (2(x^{\ell/k} + 1)^2 - 8 + (x^{\ell/k} - 1)^2(x^{\ell/12}(x^{\ell/6} + x^{\ell/12} - 1)^2))/8 \\ p(x) - 1 =& (2(x^{\ell/k} - 1)^2 + 8x^{\ell/k} - 8 + (x^{\ell/k} - 1)^2(x^{\ell/12}(x^{\ell/6} + x^{\ell/12} - 1)^2))/8 \\ p(x) - 1 =& (x^{\ell/k} - 1)(8 + (x^{\ell/k} - 1)(2 + x^{\ell/12}(x^{\ell/6} + x^{\ell/12} - 1)^2)/8 \end{aligned}$$

It is straightforward to see that $(x^{\ell/k} - 1)/2$ divides $y(x)$. □

From Table 3.13 and Theorem 3.1, we obtain Theorem 3.4.

Theorem 3.4. *For construction 6.7, let $\ell = \mathrm{lcm}(k, 8)$. The curve cofactor has a factor $c_0(x) = (x^{\ell/k} - 1)^2/8$, whose structure is $\mathbb{Z}/(x^{\ell/k} - 1)/4\mathbb{Z} \times \mathbb{Z}/(x^{\ell/k} - 1)/2\mathbb{Z}$, and it is enough to multiply by $n(x) = (x^{\ell/k} - 1)/2$ to clear the co-factor $c_0(x)$.*

Other constructions

We also investigated the KSS curves named Constructions 6.11, 6.12, 6.13, 6.14, 6.15 in [FST10], and the KSS-54 curve of 2018, but none of the cofactors is a square, and the gcd of the cofactor and $p(x)-1$, resp. $y(x)$, is equal to 1. Hence our faster co-factor clearing does not apply. We also briefly looked into the Aurifeuillean construction [SG18] for which it dos not apply neither.

3.2.2 Faster subgroup membership testing

For completeness, we first state the previously known membership tests for $\mathbb{G}_1$ [Sco21, Bow19]. Then we show our results for $\mathbb{G}_T$ [EHG22]. Next, we show that the proof argument for the $\mathbb{G}_2$ test in [Sco21] is incomplete and provide a fix and a generalization for both $\mathbb{G}_1$ and $\mathbb{G}_2$.

For the sequel, we recall that the curves of interest have a j-invariant 0 and are equipped with efficient endomorphisms ϕ on $\mathbb{G}_1$ and ψ on $\mathbb{G}_2$ (cf. Sec. 1.2).

$\mathbb{G}_1$ membership testing

Given a point $P \in E(\mathbb{F}_p)$, Scott [Sco21, §6] proves by contradiction that for BLS12 curves it is sufficient to verify that $\phi(P) = -u^2P$ where $-u^2$ is the eigenvalue λ of ϕ. A similar test was already proposed in a preprint by Bowe [Bow19, §3.2] for the BLS12-381: $((u^2-1)/3)(2\phi'(P) - P - \phi'^2(P)) - \phi'^2(P) = \mathcal{O}$ (where ϕ' here is ϕ^2). This boils down to exactly $\phi(P) = -u^2P$ using $\phi^2(P) + \phi(P) + P = \mathcal{O}$ and $\lambda^2 + \lambda + 1 \equiv 0 \mod r$ ($u^4 \equiv u^2 - 1 \mod r$). However, the proof uses a tautological reasoning, as reproached by Scott [Sco21, footnote p. 6], because it replaces λP by $\phi(P)$ where P is a point yet to be proven of order r.

$\mathbb{G}_T$ membership testing

Testing membership in $\mathbb{G}_T$ for candidate elements z of $\mathbb{F}_{p^k}$ is done in two steps. First, one checks that z belongs to the *cyclotomic subgroup* of $\mathbb{F}_{p^k}$ (subgroup of order $\Phi_k(p)$), that is $z^{\Phi_k(p)} = 1$. To avoid inversions, one multiplies the positive terms in p^i on one hand, and the negative terms on the other hand, and check for equality: it costs only Frobenius powers. With $k = 6$ and $\Phi_6(p) = p^2 - p + 1$, it means checking that $z^{p^2+1} = z^p$. Second, we propose to use a generalisation of Scott's technique first developed for BN curves, where $r = p+1-t$ [Sco13, §8.3]. In the BN case, the computation of z^r is replaced by a Frobenius power z^p and an exponentiation z^{t-1}, and the test $z^p = z^{t-1}$. BLS curves are not of prime order, and we use Proposition 3.1.

Proposition 3.1. *Let E be a pairing-friendly curve defined over $\mathbb{F}_p$, of embedding degree k w.r.t. the subgroup order r, and order $\#E(\mathbb{F}_p) = r \cdot c = p + 1 - t$. For $z \in \mathbb{F}_{p^k}^*$, we have this alternative $\mathbb{G}_T$ membership testing:*

$$z^{\Phi_k(p)} = 1 \text{ and } z^p = z^{t-1} \text{ and } \gcd(p + 1 - t, \Phi_k(p)) = r \implies z^r = 1 \ .$$

Proof. If $z^{\Phi_k(p)} = 1$ and $z^{p+1-t} = 1$, then the order of z divides the gcd of the exponents $\gcd(\Phi_k(p), p + 1 - t)$. If this gcd is exactly r, then z is in the subgroup of order r, that is $z^r = 1$. □

For example, BLS curves have $c \cdot r = p + 1 - t = p - u$ hence

$$p \equiv u \bmod r \ .$$

As soon as $\gcd(p + 1 - t, \Phi_k(p)) = r$, then the following two tests are enough:

1. test if $z^{\Phi_k(p)} = 1$ with Frobenius maps;
2. test if $z^p = z^u$, using cyclotomic squarings [GS10] for a faster exponentiation.

Remark 3.1. *For BLS-curves of embedding degree k a power of 3 ($k = 3^j$), the cyclotomic polynomial $\Phi_k(x)$ does not generate primes, actually one has $r(x) = \Phi_k(x)/3$. Moreover a BLS curve has points of order 3, hence $\gcd(p + 1 - t, \Phi_k(p)) = 3r$ for all $k = 3^j$.*

$\mathbb{G}_2$ membership

Following [Sco21, Section 4], let $E(\mathbb{F}_p)$ be an elliptic curve of j-invariant 0 and embedding degree $k = 12$. Let E' be the sextic twist of E defined over $\mathbb{F}_{p^{k/d}} = \mathbb{F}_{p^2}$, and ψ the "untwist-Frobenius-twist" endomorphism with the minimal polynomial

$$\chi(X) = X^2 - tX + p \ . \tag{3.7}$$

Let $Q \in E'(\mathbb{F}_{p^2})$. We have $\gcd(p + 1 - t, \#E'(\mathbb{F}_{p^2})) = r$. To check if Q is in $E'(\mathbb{F}_{p^2})[r]$, it is therefore sufficient to verify that

$$[p + 1 - t]Q = \mathcal{O} \ .$$

Since $[p] = -\psi^2 + [t] \circ \psi$ from Eq. (3.7), the test to perform becomes

$$\psi \circ ([t]Q - \psi(Q)) + Q - [t]Q = 0 \ . \tag{3.8}$$

It is an efficient test since ψ is fast to evaluate and $[t]Q$ can be computed once and cheaper than $[r]Q$. For BLS12 curves $t = u+1$ and the test to perform becomes in [Sco21, Section 4] the quadratic equation

$$\psi(uQ) + \psi(Q) - \psi^2(Q) = uQ \ .$$

So far, the only used fact is $\chi(\psi) = 0$, which is true everywhere. So the reasoning is correct and we have

$$\psi(uQ) + \psi(Q) - \psi^2(Q) = uQ \implies Q \in E'(\mathbb{F}_{p^2})[r] \ .$$

However the preprint [Sco21, Section 4] goes further and writes that the quadratic equation has only two solutions, $\psi(Q) = Q$ and $\psi(Q) = uQ$. Since ψ does not act trivially on $E'(\mathbb{F}_{p^2})$ the conclusion is

$$\psi(Q) = uQ \implies Q \in E'(\mathbb{F}_{p^2})[r] \ . \tag{3.9}$$

The issue. The previous property is, by luck, true as we will show later (Example 3.3). However, the overall reasoning is flawed, because it circles back to the fact that ψ acts as the multiplication by u on $\mathbb{G}_2$, while we are trying to prove that Q is in $\mathbb{G}_2$. This is the same kind of tautological reasoning reproached in the footnote of Scott's preprint [Sco21]. This reasoning implicitly supposes ψ acts as the multiplication by u only on $E'(\mathbb{F}_{p^2})[r]$, and therefore that this action characterizes $E'(\mathbb{F}_{p^2})[r]$. However, $E'(\mathbb{F}_{p^2})[r]$ might not be the only subgroup of $E'(\mathbb{F}_{p^2})$ on which ψ has the eigenvalue u. Indeed, if a prime number ℓ divides the cofactor c_2 and $\chi(u) = 0 \bmod \ell$, it is possible that, on $E'(\mathbb{F}_{p^2})[\ell]$, ψ acts as the multiplication by u, for instance if $E'(\mathbb{F}_{p^2})[\ell]$ contains the eigenspace associated to u. So the implication (3.9) is true, provided that no such prime exists.

The solution. The implication (3.9) becomes true if we know that there is no other subgroup of $E'(\mathbb{F}_{p^2})$ on which ψ acts as the multiplication by u. To make sure of this, it is enough to check that $\chi(u) \neq 0 \bmod \ell_i$ for all primes ℓ_i dividing c_2. If that is the case, we know that ψ acts as the multiplication by u only on $E'(\mathbb{F}_{p^2})[r]$. Using the Chinese Remainder Theorem it gives the following criterion:

Proposition 3.2. *If ψ acts as the multiplication by u on $E'(\mathbb{F}_{p^2})[r]$ and $\gcd(\chi(u), c_2) = 1$ then*

$$\psi(Q) = [u]Q \implies Q \in E'(\mathbb{F}_{p^2})[r] \ .$$

Note that checking the gcd of the polynomials $\chi(\lambda(X))$ and $c_2(X)$ is not sufficient and one needs to check the gcd of the integers, that are evaluations of the polynomials at u. In fact, $\gcd(\chi(\lambda(X)), c_2(X)) = 1$ in $\mathbb{Q}[X]$ only means that there is a relation $A\chi(\lambda)+Bc_2 = 1$ where $A, B \in \mathbb{Q}[X]$. The seeds u are chosen so that $\chi(\lambda(u)), c_2(u)$ are integers, but it might not be the case for $A(u)$ and $B(u)$. If d is the common denominator of the coefficients of A and B, we can only say that for a given seed u, $\gcd(\chi(u), c_2(u)) \mid d$. Therefore, we have to take care of the "exceptional seeds" u such that $\gcd(\chi(u), c_2(u))$ is a proper divisor of d.

A generalisation of $\mathbb{G}_1$ and $\mathbb{G}_2$ membership tests

Proposition 3.2 can be generalized to both $\mathbb{G}_1$ and $\mathbb{G}_2$ groups for any polynomial-based family of elliptic curves (e.g. BLS, BN, KSS). Let $\tilde{E}(\mathbb{F}_{\tilde{p}})$ be a family of elliptic curves (i.e. it can be $E(\mathbb{F}_p)$ or $E'(\mathbb{F}_{p^{k/d}})$ for instance). Let $\mathbb{G}$ be a cryptographic group of $\tilde{E}$ of order r equipped with an efficient endomorphism $\tilde{\phi}$. It has a minimal polynomial $\tilde{\chi}$ and an eigenvalue $\tilde{\lambda}$. Let c be the cofactor of $\mathbb{G}$. Proposition 3.2 becomes then

Proposition 3.3. *If $\tilde{\phi}$ acts as the multiplication by $\tilde{\lambda}$ on $\tilde{E}(\mathbb{F}_{\tilde{p}})[r]$ and $\gcd(\tilde{\chi}(\tilde{\lambda}), c) = 1$ then*

$$\tilde{\phi}(Q) = [\tilde{\lambda}]Q \implies Q \in \tilde{E}(\mathbb{F}_{\tilde{p}})[r] \ .$$

Example 3.2 (BN)**.** *Let $E(\mathbb{F}_{p(x)})$ define the BN pairing-friendly family. It is parameterized by*

$$\begin{aligned} p(x) &= 36x^4 + 36x^3 + 24x^2 + 6x + 1 \\ r(x) &= 36x^4 + 36x^3 + 18x^2 + 6x + 1 \\ t(x) &= 6x^2 + 1 \end{aligned}$$

and $E(\mathbb{F}_{p(x)})$ has a prime order so $c_1 = 1$. The cofactor on the sextic twist $E'(\mathbb{F}_{p^2})$ is $c = c_2$

$$\begin{aligned} c_2(x) &= p(x) - 1 + t(x) \\ &= 36x^4 + 36x^3 + 30x^2 + 6x + 1 \ . \end{aligned}$$

On $\mathbb{G} = \mathbb{G}_2 = E'(\mathbb{F}_{p^2})[r]$, $\tilde{\phi} = \psi$ (the "untwist-Frobenius-twist") has a minimal polynomial $\tilde{\chi} = \chi$ and an eigenvalue $\tilde{\lambda} = \lambda$

$$\begin{aligned} \chi &= X^2 - tX + p \\ \lambda &= 6X^2 \ . \end{aligned}$$

We have $\gcd(c_2, \chi(\lambda)) = \gcd(c_2(X), \chi(6X^2)) = 1$, and running the extended Euclidean algorithm we find a relation $Ac_2 + B\chi(\lambda) = 1$ where $A, B \in \mathbb{Q}[X]$. The common denominator of the coefficients of A and B is $d = 2$. We now look at the congruence relations the seed u should satisfy so that $\chi(\lambda(u))$ and $c_2(u)$ are both divisible by 2: those will be the exceptional seeds, under which the implication (3.9) could be false. Since c_2 is always odd there is no exceptional seeds and we obtain:

Proposition 3.4. *For the BN family, if $Q \in E'(\mathbb{F}_{p^2})$,*

$$\psi(Q) = [u]Q \implies Q \in E'(\mathbb{F}_{p^2})[r] \ .$$

Example 3.3 (BLS12)**.** *The BLS12 parameters are:*

$$\begin{aligned} p(x) &= (x-1)^2/3 \cdot r(x) + x \\ r(x) &= x^4 - x^2 + 1 \\ t(x) &= x + 1 \ . \end{aligned}$$

On $\mathbb{G} = \mathbb{G}_1 = E(\mathbb{F}_p)[r]$*, the endomorphism* $\tilde{\phi} = \phi$ *has minimal polynomial* $\tilde{\chi} = \chi$ *and eigenvalue* $\tilde{\lambda} = \lambda$ *as follows:*

$$\begin{aligned} \chi &= X^2 + X + 1 \\ \lambda &= -X^2 \ . \end{aligned}$$

We have $c = c_1 = (X-1)^2/3$*. Running the extended Euclidean algorithm on* c_1 *and* $\chi(\lambda)$*, we find a relation* $Ac_1 + B\chi(\lambda) = 1$ *in* $\mathbb{Q}[X]$*. In fact, here* A *and* B *are in* $\mathbb{Z}[X]$*, so there are no exceptional cases: for any acceptable seed* u*,* $\gcd(c_1(u), \chi(\lambda(u))) = 1$*, so we retrieve the result from Scott's paper [Sco21]:*

Proposition 3.5. *For the BLS12 family, if* $Q \in E(\mathbb{F}_p)$*,*

$$\phi(Q) = [-u^2]Q \implies Q \in E(\mathbb{F}_p)[r] \ .$$

On $\mathbb{G} = \mathbb{G}_2 = E'(\mathbb{F}_{p^2})[r]$*,* $\tilde{\phi} = \psi$ *(the "untwist-Frobenius-twist") has a minimal polynomial* $\tilde{\chi} = \chi$ *and an eigenvalue* $\tilde{\lambda}$*, where*

$$\begin{aligned} \chi &= X^2 - tX + p \\ \lambda &= X \ . \end{aligned}$$

The $\mathbb{G}_2$ *cofactor is* $c = c_2$

$$c_2(x) = (x^8 - 4x^7 + 5x^6 - 4x^4 + 6x^3 - 4x^2 - 4x + 13)/9 \ .$$

We have $\gcd(c_2, \chi(\lambda)) = 1$ *and running the extended Euclidean algorithm we find a relation* $Ac_2 + B\chi(\lambda) = 1$ *where* $A, B \in \mathbb{Q}[X]$*. The common denominator of the coefficients of* A *and* B *is* $3 \cdot 181$*. We look at what congruence properties the seed* u *should have so that* $\chi(\lambda(u))$ *and* $c_2(u)$ *are both divisible by* 181 *or* 3 *to rule out the exceptional cases (as before, with those seeds, the implication* (3.9) *could be false). We find that there is no seed* u *such that* $3 \mid c_2(u)$*. Furthermore, the seeds* u *such that* $181 \mid \chi(\lambda(u))$ *and* $181 \mid c_2(u)$ *are such that* $u \equiv 7 \bmod 181$ *and in that case,* $181 \mid r(u)$*. Therefore there are no exceptional cases as long as* r *is prime, and we obtain:*

Proposition 3.6. *For the BLS12 family, if* $r = r(u)$ *is prime and* $Q \in E'(\mathbb{F}_{p^2})$*,*

$$\psi(Q) = [u]Q \implies Q \in E'(\mathbb{F}_{p^2})[r] \ .$$

Example 3.4 (BLS24)**.** *The BLS24 family is parameterized by*

$$\begin{aligned} p(x) &= (x-1)^2/3 \cdot r(x) + x \\ r(x) &= x^8 - x^4 + 1 \\ t(x) &= x + 1 \ . \end{aligned}$$

On $\mathbb{G} = \mathbb{G}_1 = E(\mathbb{F}_p)[r]$, the endomorphism $\tilde{\phi} = \phi$ has minimal polynomial $\tilde{\chi} = \chi$ and eigenvalue $\tilde{\lambda} = \lambda$, where

$$\begin{aligned}\chi &= X^2 + X + 1\\ \lambda &= -X^4 \ .\end{aligned}$$

We have $c = c_1 = (X-1)^2/3$. Running the extended Euclidean algorithm on c_1 and $\chi(\lambda)$, we find a relation $Ac_1 + B\chi(\lambda) = 1$ in $\mathbb{Q}[X]$. As for BLS12, A and B are in $\mathbb{Z}[X]$, so there are no exceptional cases, and we have

Proposition 3.7. *For the BLS24 family, if $Q \in E(\mathbb{F}_p)$,*

$$\phi(Q) = [-u^4]Q \implies Q \in E(\mathbb{F}_p)[r] \ .$$

On $\mathbb{G} = \mathbb{G}_2 = E'(\mathbb{F}_{p^4})[r]$, $\tilde{\phi} = \psi$, the "untwist-Frobenius-twist" has a minimal polynomial $\tilde{\chi} = \chi$ and an eigenvalue $\tilde{\lambda} = \lambda$, where

$$\begin{aligned}\chi &= X^2 - tX + p\\ \lambda &= X \ .\end{aligned}$$

The cofactor on the sextic twist $E'(\mathbb{F}_{p^4})$ is $c = c_2$

$$\begin{aligned}c_2(x) =&(x^{32} - 8x^{31} + 28x^{30} - 56x^{29} + 67x^{28} - 32x^{27} - 56x^{26} + 160x^{25} - 203x^{24} + 132x^{23}\\ &+ 12x^{22} - 132x^{21} + 170x^{20} - 124x^{19} + 44x^{18} - 4x^{17} + 2x^{16} + 20x^{15} - 46x^{14} + 20x^{13}\\ &+ 5x^{12} + 24x^{11} - 42x^{10} + 48x^9 - 101x^8 + 100x^7 + 70x^6 - 128x^5 + 70x^4 - 56x^3\\ &- 44x^2 + 40x + 100)/81 \ .\end{aligned}$$

We have $\gcd(c_2, \chi(\lambda)) = 1$. Running the extended Euclidean algorithm on c_2 and $\chi(\lambda)$, we find a relation $Ac_2 + B\chi(\lambda) = 1$ where the common denominator of the coefficients of A and B is $3^5 \times 1038721$. As before, we find that there is no seed u such that $3 \mid c_2(u)$. Moreover, the seeds u such that $1038721 \mid c_2(u)$ and $1038721 \mid \chi(\lambda)$ are such that $u = 162316 \bmod 1038721$. In this case $1038721 \mid r(u)$ and hence there are no exceptional cases. We obtain:

Proposition 3.8. *For the BLS24 family, if $r = r(u)$ is prime and $Q \in E'(\mathbb{F}_{p^4})$, then*

$$\psi(Q) = [u]Q \implies Q \in E'(\mathbb{F}_{p^4})[r] \ .$$

Example 3.5 (BLS48)**.** *The BLS48 family is parametrized by*

$$\begin{aligned}p(x) &= (x-1)^2/3 \cdot r(x) + x\\ r(x) &= x^{16} - x^8 + 1\\ t(x) &= x + 1 \ .\end{aligned}$$

On $\mathbb{G} = \mathbb{G}_1 = E(\mathbb{F}_p)[r]$, the endomorphism $\tilde{\phi} = \phi$ has minimal polynomial $\tilde{\chi} = \chi$ and eigenvalue $\tilde{\lambda} = \lambda$, where

$$\begin{aligned}\chi &= X^2 + X + 1\\ \lambda &= -X^8 \ .\end{aligned}$$

We have $c = c_1 = (X-1)^2/3$. Running the extended Euclidean algorithm on c_1 and $\chi(\lambda)$, we find a relation $Ac_1 + B\chi(\lambda) = 1$ in $\mathbb{Q}[X]$. As for BLS12 and BLS24, A and B are in $\mathbb{Z}[X]$, so there are no exceptional cases, and we have

Proposition 3.9. *For the BLS48 family, if* $Q \in E(\mathbb{F}_p)$,

$$\phi(Q) = [-u^8]Q \implies Q \in E(\mathbb{F}_p)[r] \ .$$

On $\mathbb{G} = \mathbb{G}_2 = E'(\mathbb{F}_{p^8})[r]$, $\tilde{\phi} = \psi$, *the "untwist-Frobenius-twist" has a minimal polynomial* $\tilde{\chi} = \chi$ *and an eigenvalue* $\tilde{\lambda} = \lambda$, *where*

$$\begin{aligned} \chi &= X^2 - tX + p \\ \lambda &= X \ . \end{aligned}$$

The cofactor on the sextic twist $E'(\mathbb{F}_{p^8})$ *is* $c = c_2 = (p^8(x) + 1 - (3y_8(x) + t_8(x))/2)/r(x)$

$$c_2(x) = (x^{128} - 16x^{127} + 120x^{126} - 560x^{125} + \cdots + 6481)/6561 \ .$$

We have $\gcd(c_2, \chi(\lambda)) = 1$. *Running the extended Euclidean algorithm on* c_2 *and* $\chi(\lambda)$, *we find a relation* $Ac_2 + B\chi(\lambda) = 1$ *where the common denominator of the coefficients of* A *and* B *is* $1153 \times 4726299241057$. *We now look at the congruence relations the seed* u *should satisfy so that* $\chi(\lambda(u))$ *and* $c_2(u)$ *are both divisible either by 1153 or 4726299241057: Those will be the exceptional seeds, under which the implication* (3.9) *could be false. We note* U_{p_i} *the set of values of* $u \bmod p_i$ *such that* $\chi(\lambda)(x) = 0 \bmod p_i$ *and similarly* V_{p_i} *the set of values of* $u \bmod p_i$ *such that* $c_2(u) = 0 \bmod p_i$.

$$\begin{aligned} &p_i = 1153 : && U_{1153} \cap V_{1153} = \{1135\} \\ &p_i = 4726299241057 : && U_{4726299241057} \cap V_{4726299241057} = \{2085225345771\} \end{aligned}$$

For the exceptional seeds $u \equiv 1135 \bmod 1153$ *and* $u \equiv 2085225345771 \bmod 4726299241057$, *we need to check that* $\gcd(\chi(\lambda)(u), c_2(u)) = 1$ *over the integer instances (i.e. for the concrete values of* x*). However, in both cases* r *is not a prime. So we have*

Proposition 3.10. *For the BLS48 family, if* $r = r(u)$ *is prime and* $Q \in E'(\mathbb{F}_{p^8})$,

$$\psi(Q) = [u]Q \implies Q \in E'(\mathbb{F}_{p^8})[r] \ .$$

Chapter

4

Families of SNARK-friendly 2-cycles and 2-chains

In this chapter we focus on the construction of families of cycles and chains of SNARK-friendly elliptic curves for recursive proof systems. First, we present results from the literature on cycles of pairing-friendly, plain and hybrid curves before presenting our results on families of 2-chains and their arithmetic. This chapter, in part, is a reprint of the material as it appears in our published works [AEHG22] and [EHG22].

4.1 Cycles of pairing-friendly elliptic curves

A *cycle* of elliptic curves is a list of curves defined over finite fields in which the number of points in one curve is equal to the size of the field of definition of the next curve, in a cyclic manner.

Definition 4.1. *An m-cycle of elliptic curves is a list of m distinct elliptic curves*

$$E_1/\mathbb{F}_{p_1}, \ldots, E_m/\mathbb{F}_{p_m},$$

for primes $p_1, \ldots, p_m$, such that the numbers of points on these curves satisfy

$$\#E_1(\mathbb{F}_{p_1}) = p_2, \ldots, \#E_i(\mathbb{F}_{p_i}) = p_{i+1}, \ldots, \#E_m(\mathbb{F}_{p_m}) = p_1.$$

This notion was initially introduced under different names, for example *amicable pairs* (or equivalently *dual elliptic primes* [Mih07b]) for 2-cycles of ordinary curves, and *aliquot cycles* for the general case [SS11]. A *pairing-friendly m-cycle* is an m-cycle composed of ordinary pairing-friendly curves.

The state of the art.

Chiesa, Chu and Weidner studied the existence of pairing-friendly cycles, beyond the previously known cycles of curves with embedding degrees 4 and 6, and arrived at a

number of negative results which indicate that having a small embedding degree is a *strong* restriction on constructing cycles [CCW19]:

- There are no 2-cycles of elliptic curves with embedding degrees $(5, 10), (8, 8)$ or $(12, 12)$, which means that there are no *optimal* (in terms of parameter sizes) pairing-friendly 2-cycles at the 128-bit security level.
- There are no pairing-friendly cycles with more than 2 curves with the same CM discriminant $D > 3$, which implies that elliptic curves from families of varying discriminants must be used to construct cycles.
- There are no cycles of prime-order pairing-friendly curves only from the Freeman and Barreto-Naehrig families; or cycles of composite-order elliptic curves. This motivates the search for new constructions of prime-order pairing-friendly curves.

On the positive side, the authors characterize all cycles of MNT curves as consisting of curves with alternating embedding degrees of 4 or 6, and construct a new 4-cycle of MNT curves which consists of the union of 2-cycles. Table 4.1 collects the possibilities to generate MNT cycles, for which a specific cycle can be generated by substituting the possible choices of parameter x and checking that all polynomials $p(x)$ and $r(x)$ evaluate to prime numbers. Concrete parameters used in practice can be found in Table 3.3.

Open problems.

The above restrictions and current constructions of cycles suggest that pairing-friendly 2-cycles will always have the inconvenience of including two elliptic curves at different security levels, which means that at least one curve would have sub-optimal performance, but there are several *open problems* for which a satisfying solution would potentially increase the available options:

- Are there cycles of elliptic curves with the same embedding degree, and possibly the same discriminant?
- Are there pairing-friendly cycles of embedding degrees greater than 6?
- Are there pairing-friendly cycles combining MNT, Freeman and Barreto-Naehrig curves?

	(6,4,6,4) 4-cycle			
	(6,4) 2-cycle		(6,4) 2-cycle	
	E_1	E_2	E_3	E_4
k	6	4	6	4
$p(x)$	$4x^2 + 1$	$4x^2 + 2x + 1$	$4x^2 + 1$	$4x^2 - 2x + 1$
$r(x)$	$4x^2 + 2x + 1$	$4x^2 + 1$	$4x^2 - 2x + 1$	$4x^2 + 1$
$t(x)$	$-2x + 1$	$2x + 1$	$2x + 1$	$-2x + 1$

Table 4.1: Parameterized (6,4) 2-cycles and (6,4,6,4) 4-cycles of MNT curves, where 4-cycles are constructed as the union of the 2-cycles.

We already know that *optimal* 2-cycles for recursive SNARKs cannot be found, but it is possible that better alternatives with higher embedding degrees and smaller parameter sizes

than the inefficient MNT (6,4)-cycles exist. However, reconciling the conditions for pairing-friendly cycles (prime order, varying families and discriminants) with the requirements for SNARK-friendliness to construct efficient recursive SNARKs, i.e. conditions (i)–(v) (cf. page 32), further restricts the choice of parameters and increase the challenge. Since no methods are known to find prime-order pairing-friendly curves with embedding degree larger than 12, novel ideas are severely needed. In particular, finding 2-cycles of pairing-friendly curves with embedding degrees $(16, 16)$ or higher would already be a significant contribution to improving performance of constructions at current 128-bit and higher security levels.

Because of these difficulties in finding parameters, current efforts have focused on finding *hybrid cycles* containing one pairing-friendly curve, amicable pairs or *chains* of pairing-friendly elliptic curves that allow alternative recursive proofs without pairings.

4.2 Cycles of plain elliptic curves

Given the difficulties in finding optimal cycles of pairing-friendly curves (used in protocols with sub-linear verification), a recent alternative is to find amicable pairs (2-cycles) of *plain* elliptic curves, i.e., curves not equipped with efficient bilinear maps, where just the hardness of the discrete logarithm holds (used in protocols with linear verification [BGH19]). The advantages of this approach are higher performance, since parameters can scale just like traditional Elliptic Curve Cryptography, and conservativeness against advances in discrete logarithm computation over finite fields, observing that advances in solving elliptic curve discrete logarithms would impact pairing-friendly curves anyway. On the other hand, these advantages come at mild increase in proof sizes and verification times.

Silverman and Stange [SS11] showed that there exist elliptic curves with arbitrary long cycles, and conjectured that for any elliptic curve $E/\mathbb{Q}$, the number $\mathcal{Q}_E(X)$ of prime pairs (p, q) with $p < q$ and $p \leq X$ such that reducing E modulo p and q gives an amicable pair is such that:

1. $\mathcal{Q}_E(X) = \Theta\left(\dfrac{\sqrt{X}}{(\log X)^2}\right)$ as $X \to \infty$ if E does not have complex multiplication;
2. $\mathcal{Q}_E(X) \approx A_E \dfrac{X}{(\log X)^2}$ for a constant $A_E > 0$, otherwise.

In other words, there is a surprising difference between the CM and non-CM cases, and amicable pairs are asymptotically common. The reason for why CM curves have so many amicable pairs is attributed to the fact that if $E/\mathbb{Q}$ has CM and $\#E_p(\mathbb{F}_p) = q$ then there are generally only two possible values for $\#E_q(\mathbb{F}_q)$: p and $2q + 2 - p$, while non-CM curves have $\#E_q(\mathbb{F}_q)$ ranging throughout the interval given by the Hasse condition $p - 2\sqrt{p} + 1 \leq r \leq p + 2\sqrt{p} + 1$. Further research in [Jon13, Par19] has refined and proven these estimates on average.

Regarding security, one notes that the extreme case of an 1-cycle, i.e. $\#E(\mathbb{F}_p) = p$ leads to anomalous curves for which the discrete logarithm can be computed in linear time [Sma99]. For longer cycles, a set of requirements for choosing parameters can be derived from the SafeCurves [BL] criteria for ECC security:

1. Hardness of the ECDLP against Pollard's rho method on the curve and twist.
2. Large embedding degree and CM discriminant D.

3. Rigidity for generating parameters in a reproducible manner.

4. Availability of Montgomery ladder and complete addition laws.

5. Efficient bijective maps (encodings) between uniform random strings and a large set of rational points on the curve. Such an encoding can also be used for hashing efficiently to the curve, as required by some protocols.

A conservative choice of parameters attempts to finds curves that maximize the set of satisfied security requirements.

`sec(p|q)256k1` curves.

Probably the earliest 2-cycle of plain elliptic curves with application in zero-knowledge proofs was based on the elliptic curve adopted for signatures in Bitcoin, the `secp256k1` curve from the 2010 SEC2 standard [Bro10]. Later, the `secq256k1` was derived with the motivation to be able to build zero-knowledge proofs over already deployed cryptosystems by using the closely related curve obtained with just swapping base field and order [Poe18]:

$$
\begin{aligned}
&E_p/\mathbb{F}_p : y^2 = x^3 + 7 \text{ of order } q, \text{ called } \texttt{secp256k1} \\
&E_q/\mathbb{F}_q : y^2 = x^3 + 7 \text{ of order } p, \text{ called } \texttt{secq256k1}, \text{ with} \\
&\quad p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \text{ and} \\
&\quad q = 2^{256} - 432420386565659656852420866394968145599.
\end{aligned}
$$

These curves satisfy SafeCurves criteria 1 and 3, but have CM discriminant $D = -3$ for the efficient endomorphism [GLV01]; and prime order which does not allow the original Montgomery ladder or Elligator encoding [BHKL13]. However, an optimized version of the Elligator Squared [Tib14] encoding can be used [Wui21].

Tweedle curves.

One of the first 2-cycles of elliptic curves suitable for recursive proofs was the Tweedledum-Tweedledee curves in Halo [BGH19]:

$$
\begin{aligned}
&E_p/\mathbb{F}_p : y^2 = x^3 + 5 \text{ of order } q, \text{ called Tweedledum;} \\
&E_q/\mathbb{F}_q : y^2 = x^3 + 5 \text{ of order } p, \text{ called Tweedledee, with} \\
&\quad p = 2^{254} + 4707489545178046908921067385359695873 \text{ and} \\
&\quad q = 2^{254} + 4707489544292117082687961190295928833 4.
\end{aligned}
$$

These curves satisfy some of the SafeCurves criteria at the 126-bit security, but have CM discriminant $D = 3$ as a consequence of how they were constructed, and do not benefit from the Montgomery ladder as prime-order curves. On the other hand, they have high 2-adicity, are equipped with fast endomorphisms for efficient scalar multiplication, and somewhat efficient encoding/hashing algorithms.

Pasta curves.

Like the Tweedle curves, Pallas and Vesta form a 2-cycle of elliptic curves for recursive proofs in Halo 2 and beyond [Hop20]:

$$E_p/\mathbb{F}_p : y^2 = x^3 + 5 \text{ of order } q, \text{ called Pallas;}$$
$$E_q/\mathbb{F}_q : y^2 = x^3 + 5 \text{ of order } p, \text{ called Vesta, with}$$
$$p = 2^{254} + 45560315531419706090280762371685220353 \text{ and}$$
$$q = 2^{254} + 45560315531506369815346746415080538113.$$

These curves provide several of the attractive properties of the Tweedle curves, while relaxing twist security to 100 bits for Pallas, and otherwise improving over them on a number of ways. Both curves have:

- Lower-degree isogenies (3 instead of 23) from curves with a nonzero j-invariant, which accelerates hashing using the Wahby-Boneh method [WB19].
- The same 2-adicity of 32 (instead of 33 and 34) that accelerates square root extraction and simplifies point compression.
- Sparse moduli to accelerate base field arithmetic in the Montgomery representation.
- A reproducible generation method based on a search utility that is publicly available and should facilitate searching for similar 2-cycles in the future, satisfying rigidity concerns.

The generation method specializes the CM method for $D = 3$ to produce curves of the form $E(\mathbb{F}) : y^2 = x^3 + b$ with $b \neq 0$. It looks for values (y, t) within a certain range such that $p = (3y^2 + t^2)/4$ is a prime with the desired length and $p \equiv 1 \pmod 6$ to equip the curve with a fast endomorphism. By requiring $(y - 1)/2$ and $(t - 1)/2$ to be multiples of 2^L, $p - 1$ will be a multiple of 2^L and $q - 1$ will be a multiple of 2^L for $q \in \{p + 1 - t, p + 1 + (t - 3y)/2\}$ among the six possible orders given by the CM method. Algorithm 4.1 summarizes the generation strategy.

Algorithm 4.1: FindAmicablePair(ℓ, L)
Input: designed length ℓ for prime p, 2-adicity L, k-bit security
Output: 2-cycle of curves E_p, E_q.

while *true*:
  Find (y, t) such that:
    • $4p = 3y^2 + t^2$, $2^L \mid (y - 1)/2$, $2^L \mid (t - 1)/2$;
    • p is prime, $|p| = \ell \wedge p \equiv 1 \pmod 6$;
  for $q \in \{p + 1 - t, p + 1 + (t - 3y)/2\}$:
    if $q \notin \{p, p + 1, p - 1\} \wedge q \equiv 1 \pmod 6 \wedge 2^L | (q - 1) \wedge q$ *is prime*:
      Find b_p incrementally such that $E_p(\mathbb{F}_p) : y^2 = x^3 + b_p$ has order q;
      Find b_q incrementally such that $E_q(\mathbb{F}_q) : y^2 = x^3 + b_q$ has order p;
      if *ECDLPSecurity*$(E_p) \geq k \wedge$ *ECDLPSecurity*$(E_q) \geq k$:
        return (E_p, E_q);

Generalizing the method for constructing 2-cycles described above gives the immediate corollary that every curve generated with the CM method using a small discriminant

forms a cycle with another CM curve generated with the prescribed order given by the characteristic of the base field. In particular, if $q = p + 1 - t$ with CM discriminant D (such that $t^2 - 4p = y^2 D$), then taking $s = 2 - t$ automatically gives $p = q + 1 - s$ and $s^2 - 4q = t^2 - 4p = y^2 D$.

4.3 Hybrid cycles of elliptic curves

A middle-ground alternative to cycle of pairing-friendly curves and cycles of plain elliptic curves is *half-pairing cycles*, where only one of the curves in a 2-cycle is pairing-friendly. This can be potentially useful to combine recursive proofs with protocols based on cryptographic pairings.

Using the results from previous sections, it is possible to generate half-pairing cycles where the pairing-friendly curve is of prime order by just using the CM method. Hence this strategy applies to generating 2-cycles where the pairing-friendly curve is either from the BN, Freeman or MNT families. One such example is the Pluto-Eris curve.

Pluto-Eris curves.

Pluto and Eris form a 2-cycle of elliptic curves [Hop21]:

$$
\begin{aligned}
&E_p/\mathbb{F}_p : y^2 = x^3 + 57 \text{ is a BN curve of order } q, \text{ called Pluto;}\\
&E_q/\mathbb{F}_q : y^2 = x^3 + 57 \text{ is a plain curve of order } p, \text{ called Eris, with}\\
&\quad p = 36x^4 + 36x^3 + 24x^2 + 6x + 1 \text{ and}\\
&\quad q = 36x^4 + 36x^3 + 18x^2 + 6x + 1, \text{for } x = -(2^{110} + 2^{60} + 2^{39} + 2^{35} - 2^{31}).
\end{aligned}
$$

The field size of 446 bits for Pluto is chosen to satisfy 128 bits of security for pairings [Gui20], although it naturally leaves a larger security margin for Eris. Regarding performance, both curves have the high 2-adicity of 32, are again equipped with fast endomorphisms for efficient scalar multiplication due to the choice of small CM discriminant and efficient encoding/hashing algorithms. The choice of seed for the BN curve gives a particularly fast pairing computation due to its low Hamming weight.

Another such example is the hybrid cycle based on a BN382 curve that was briefly used by the Mina testnet in early 2020 [Mec20].

BN382-Plain curves.

BN382 forms, with a related plain curve, a 2-cycle of elliptic curves [Mec20]:

$$
\begin{aligned}
&E_p/\mathbb{F}_p : y^2 = x^3 + 14 \text{ is a BN curve of order } q;\\
&E_q/\mathbb{F}_q : y^2 = x^3 + 7 \text{ is a plain curve of order } p, \text{ with}\\
&\quad p = 36x^4 + 36x^3 + 24x^2 + 6x + 1 \text{ and}\\
&\quad q = 36x^4 + 36x^3 + 18x^2 + 6x + 1, \text{for } x = 2^{94} + 2^{81} + 2^{74} + 2^{66}.
\end{aligned}
$$

The field sizes of 382 bits are chosen to satisfy slightly less than 128 bits of security for pairings, while allowing an efficient 6-limb arithmetic on 64-bit architectures. Similarly to Pluto-Eris, both curves have the high 2-adicity of 64, are equipped with fast endomorphisms for efficient scalar multiplication due to the choice of small CM discriminant and efficient encoding/hashing algorithms. The choice of seed for the BN curve gives a particularly fast pairing computation due to its low Hamming weight.

4.4 Chains of pairing-friendly elliptic curves

We have seen in previous sections that constructing a *cycle* of two pairing-friendly curves is a very difficult task, and results in a very slow pairing computation, and large parameters. Relaxing the conditions on the curves, one can look for a chain of pairing-friendly elliptic curves, as in Definition 4.3.

Definition 4.2. *An m-chain of elliptic curves is a list of distinct curves*

$$E_1/\mathbb{F}_{p_1}, \ldots, E_m/\mathbb{F}_{p_m}$$

where $p_1, \ldots, p_m$ *are large primes and*

$$p_1 \mid \#E_2(\mathbb{F}_{p_2}), \ldots, \ p_{i-1} \mid \#E_i(\mathbb{F}_{p_i}), \ldots, \ p_{m-1} \mid \#E_m(\mathbb{F}_{p_m}) \ . \tag{4.1}$$

Definition 4.3. *An m-chain of SNARK-friendly elliptic curves is an m-chain where each of the* $\{E_i/\mathbb{F}_{p_i}\}_{1\leq i\leq m}$ *curves*

- *is pairing-friendly;*
- *has a highly 2-adic subgroup, i.e.* $r_i - 1 \equiv 0 \mod 2^L$ *for a large* $L \geq 1$ *(*$r_i \mid \#E_i(\mathbb{F}_{p_i})$*).*

In a 2-chain, the first curve is denoted the **inner curve**, while the second curve whose order is the characteristic of the inner curve, is denoted the **outer curve**.

Previous work on chains of pairing-friendly curves.

Approximately at the same time as [BCTV14a] released the first cycle of MNT curves, Costello et al. [CFH+15] presented a 2-chain of pairing-friendly curves to obtain a bounded recursive proof composition (of depth 2). Their inner curve is the popular BN curve from [NAS+08] with seed $x = -(2^{62} + 2^{55} + 1)$ and their outer curve is a new Brezing–Weng curve of embedding degree 6 and 509 bits (just under the machine-word-aligned size of 512 bits). However, the curves do not satisfy Condition (i) that is, no large power of 2 divides neither $r - 1$ nor $p - 1$. It is surprising that they did not use another widespread BN curve with seed $x = 2^{62} - 2^{54} + 2^{44}$ for example (Beuchat et al. [BGM+10], in the TEPLA library), which does satisfy Condition (i): $2^{45} \mid r - 1$ and $2^{45} \mid p - 1$. Nevertheless, the Geppetto construction is the first example of a 2-chain of pairing-friendly curves (with parameters having a polynomial form) that we found in the literature, to the best of our knowledge.

Probably because Geppetto appeared *before* the new TNFS algorithm and the re-evaluation of all key sizes of pairing-friendly curves, the MNT cycle of 298 bits was a milestone construction, while the Geppetto 2-chain was not remembered in the subsequent construction named ZEXE [BCG+20]. In light of the need of larger key-sizes, Bowe et al. released a 2-chain of pairing-friendly curves based on the BLS12 family. The inner curve is a BLS12-377 curve with $2^{46} \mid r - 1$ and $2^{46} \mid p - 1$. However the outer curve was a Cocks–Pinch curve (CP6-782) that did not benefit from all possible speed-ups of pairing-friendly curves.

In [EHG20], we introduced a 2-chain of curves made of the previous BLS12-377 and a new BW6-761 curve, a Brezing–Weng curve of embedding degree 6 defined over a 761-bit

prime field, which we demonstrated to be orders of magnitude faster than CP6-782. Indeed many optimizations are now available, from a twist of degree 6 to an optimal ate pairing, and 761 bits fit in one less machine word of 64 bits compared to 782 bits. We investigate many different possibilities of 2-chains with BLS12, BLS24 and BN inner curves, and Brezing–Weng outer curves of embedding degree 6, but also Cocks–Pinch curves of the larger embedding degrees 8 and 12.

4.4.1 Inner curves: Barreto–Lynn–Scott (BLS) curves

We investigate the BLS family as an option for a SNARK-friendly inner curve. We first present our results for a better arithmetic on all BLS curves and then argue on the choice of BLS12 and BLS24 curves for our applications. Later in the survey paper [AEHG22], we generalize the framework to include BN curves as well.

Parameters with a polynomial form

BLS curves were introduced in [BLS03]. This is a family of pairing-friendly elliptic curves of embedding degree k multiple of 3 but not multiple of 18. Well-known families are given with $k = 2^i 3^j$ for $i, j \geq 0$: $k = 9, 12, 24, 27, 48$ (Table 7.1). The curves have j-invariant 0, discriminant $-D = -3$. Each family has polynomial parameters $p(x), r(x), t(x)$ for characteristic, subgroup order of embedding degree k, and trace. The subgroup order is $r(x) = \Phi_k(x)$ the k-th cyclotomic polynomial. The trace has a simple expression $t(x) = x + 1$, so that the ate pairing whose Miller loop computes the function $f_{x,Q}(P)$ is optimal in terms of Vercauteren's paper [Ver10]. The curve order is $p(x) + 1 - t(x)$ and the CM equation is $4p(x) = t(x)^2 + Dy(x)^2$. We state below useful lemmas and their proofs. The explicit polynomials for BLS curves with $k \leq 99$ are given in Tables 4.3 and 4.4.

Lemma 4.1. *The cofactor $c_1(x)$ of BLS curves such that $p(x) + 1 - t(x) = c_1(x)r(x)$ has the form*

1. *$(x-1)^2/3 \cdot c_2(x)$ for odd k, where $c_2(x) = (x^{2k/3} + x^{k/3} + 1)/\Phi_k(x) \in \mathbb{Q}[x]$;*
2. *$(x-1)^2/3 \cdot c_2(x)$ for even k, where $c_2(x) = (x^{k/3} - x^{k/6} + 1)/\Phi_k(x) \in \mathbb{Q}[x]$.*

Lemma 4.2. *For all BLS curves, the polynomial form of the characteristic $p(x)$ is such that $(x-1)/3$ divides $p(x) - 1$.*

Proof (of Lemma 4.2). Observe that $q(x) - 1 = c(x)r(x) + t(x) - 2$, and $t(x) - 2 = x - 1$. For odd k, from Lemma 4.1 one has $q(x) - 1 = (x-1)^2/3 \cdot (x^{2k/3} + x^{k/3} + 1) + x - 1 = (x-1)/3 \cdot ((x-1)(x^{2k/3} + x^{k/3} + 1) + 1)$.

For even k, from Lemma 4.1 one has $q(x) - 1 = (x-1)^2/3 \cdot (x^{k/3} - x^{k/6} + 1) + x - 1 = (x-1)/3 \cdot ((x-1)(x^{k/3} - x^{k/6} + 1) + 1)$. In both cases, $(x-1)/3$ divides $q(x) - 1$. □

Lemma 4.3. *The parameter $y(x)$ of BLS curves has the form*

1. *$(x-1)(2x^{k/3} + 1)/3$ for odd k;*
2. *$(x-1)(2x^{k/6} - 1)/3$ for even k.*

Proof (of Lemmas 4.1 and 4.3). Let us consider odd k. Observe that $x^{k/3}$ is a primitive third root of unity $(-1 + \sqrt{-3})/2$ modulo $r(x) = \Phi_k(x)$, and $1/\sqrt{-3} = (2x^{k/3} + 1)/3$. A solution for $y(x) = (t(x) - 2)/\sqrt{-3} \bmod r(x)$ is $y(x) = (x-1)(2x^{k/3} + 1)/3$, an then

Table 4.2: Parameters of BLS curves for $k = 2^i 3^j$, $i \geq 0$, $j \geq 1$, $18 \nmid k$.

k	$2^i 3^j,\ i,j \geq 1\ (6, 12, 24, 48, 96, \ldots)$	$3^j,\ j \geq 1\ (3, 9, 27, 81, \ldots)$
$t(x)$	$x+1$	
$y(x)$	$(x-1)(2x^{k/6}-1)/3$	$(x-1)(2x^{k/3}+1)/3$
$r(x)$	$x^{k/3}-x^{k/6}+1$	$x^{2k/3}+x^{k/3}+1$
$p(x)$	$r(x)(x-1)^2/3+x$	$r(x)/3(x-1)^2+x$
$c_2(x)$	1	1
ρ	$1+6/k$	$1+3/k$

$q(x) = (t^2(x)+3y^2(x))/4$ is an irreducible polynomial which represents primes in the terms of [FST10, Definition 2.5]. The curve order is $q(x)+1-t(x) = ((t(x)-2)^2+3y^2(x))/4 = ((x-1)^2+(x-1)^2(2x^{k/3}+1)^2/3)/4 = (x-1)^2/3(x^{2k/3}+x^{k/3}+1) = c(x)r(x)$. Note that $x^k-1 = (x^{2k/3}+x^{k/3}+1)(x^{k/3}-1)$, hence $\Phi_k(x)$ divides $x^{2k/3}+x^{k/3}+1$ (as it does not divide $x^{k/3}-1$), and the cofactor $c(x)$ has the form

$$c(x) = (x-1)^2/3 \cdot (x^{2k/3}+x^{k/3}+1)/\Phi_k(x)\ .$$

In particular for $k = 3^j$, the k-th cyclotomic polynomial is $\Phi_{3^j}(x) = \Phi_3(x^{3^{j-1}}) = x^{2k/3}+x^{k/3}+1$, in this case the cofactor $c(x)$ is exactly $(x-1)^2/3$.

With even k, $x^{k/6}$ is a primitive 6-th root of unity $(1+\sqrt{-3})/2$ modulo $r(x) = \Phi_k(x)$, and $1/\sqrt{-3} = (2x^{k/6}-1)/3$. Then $y(x) = (x-1)(2x^{k/6}-1)/3$, and $q(x) = (t^2(x)+3y^2(x))/4$ is an irreducible polynomial which represents primes in the terms of [FST10, Definition 2.5]. The curve order is $q(x)+1-t(x) = ((t(x)-2)^2+3y^2(x))/4 = ((x-1)^2+(x-1)^2(2x^{k/6}-1)^2/3)/4 = (x-1)^2/3(x^{k/3}-x^{k/6}+1) = c(x)r(x)$. In the same way as for odd k, one observes that $x^k-1 = (x^{k/3}-x^{k/6}+1)(x^{k/3}+x^{k/6}+1)(x^{k/3}-1)$, hence $\Phi_k(x)$ divides $x^{k/3}-x^{k/6}+1$, and the cofactor $c(x)$ has the form

$$c(x) = (x-1)^2/3 \cdot (x^{k/3}-x^{k/6}+1)/\Phi_k(x)\ .$$

□

Lemma 4.4. *Any BLS curve has endomorphism ring $\mathbb{Z}[\omega]$ where $\omega = (1+\sqrt{-3})/2$.*

Proof (of Lemma 4.4). Any BLS curve is an ordinary curve of j-invariant 0 and discriminant -3, of the form $y^2 = x^3+b$, defined over a prime field $\mathbb{F}_p$ where $q = 1 \bmod 3$. In this case, it is well known that the (GLV) endomorphism is of the form $\psi\colon (x,y) \mapsto (\omega x, y)$, where $\omega \in \mathbb{F}_p$ is a primitive third root of unity. It has characteristic polynomial $\psi^2+\psi+1 = 0$ and is defined over $\mathbb{F}_p$. The endomorphism ring of the curve is $\mathbb{Z}[(1+\sqrt{-3})/2]$. □

Choosing a curve coefficient $b = 1$

Proposition 4.1. *Half of BLS curves are of the form $Y^2 = X^3+1$, these are the curves with odd seed x.*

Proof. Let $E : Y^2 = X^3+b$ be a BLS curve over $\mathbb{F}_p$ and g neither a square nor a cube in $\mathbb{F}_p$. One choice of $b \in \{1, g, g^2, g^3, g^4, g^5\}$ gives a curve with the correct order (i.e. $r \mid \#E(\mathbb{F}_p)$) [Sil09, §X.5]. For all BLS curves, $x-1 \mid \#E(\mathbb{F}_p)$ (cf Lemma 4.1, Tables 4.3, 4.4) and $3 \mid x-1$ (which leads to all involved parameters being integers). If, additionally, $2 \mid x-1$ then $2, 3 \mid \#E(\mathbb{F}_p)$ and the curve has points of order 2 and 3. A 2-torsion point is $(x_0, 0)$

Table 4.3: Parameters of BLS curves, $6 \mid k$, $18 \nmid k$.

k	$6, 12, 24, 48, 96$	$30, 42, 66, 78$	$60, 84$
$t(x)$	$x+1$		
$y(x)$	$(x-1)(2x^{k/6}-1)/3$		
$r(x)$	$x^{k/3}-x^{k/6}+1$	$\Phi_k(x)$	
$q(x)$	$r(x)(x-1)^2/3+x$	$r(x)(x-1)^2/3c_2(x)+x$	
$c_2(x)$	1	x^2-x+1	x^4-x^2+1
ρ	$1+6/k$	$(k/3+2)/\varphi(k)$	

Table 4.4: Parameters of BLS curves, $k = 3 \bmod 6$.

k	$3, 9, 27, 81$	$15, 21, 33, 39,$ $51, 57, 69, 87, 93$	$45, 63, 99$	75
$t(x)$	$x+1$			
$y(x)$	$(x-1)(2x^{k/3}+1)/3$			
$r(x)$	$x^{2k/3}+x^{k/3}+1$	$\Phi_k(x)$		
$q(x)$	$r(x)/3(x-1)^2+x$	$r(x)(x-1)^2/3c_2(x)+x$		
$c_2(x)$	1	x^2+x+1	x^6+x^3+1	$x^{10}+x^5+1$
ρ	$1+3/k$	$(2k/3+2)/\varphi(k)$		

with x_0 a root of x^3+b, hence $b=(-x_0)^3$ is a cube. The two 3-torsion points are $(0, \pm\sqrt{b})$ hence b is a square. This implies that b is a square and a cube in $\mathbb{F}_p$ and therefore $b=1$ is the only solution in the set $\{g^i\}_{0\leq i\leq 5}$ for half of all BLS curves: those with odd x.

□

SNARK-friendly inner BLS curves

We focus on inner SNARK-friendly BLS curves as in Definition 4.3 at the 128-bit security level and suitable for the Groth16 and KZG-based universal SNARKs. On the one hand, a Groth16-tailored curve should optimize $\mathbb{G}_1$ and $\mathbb{G}_2$ operations, and the pairing computation: the proving algorithm involves MSMs in $\mathbb{G}_1$ and $\mathbb{G}_2$, and the verification algorithm involves multi-pairings. On the other hand, KZG polynomial commitments only need multi-scalar multiplications in $\mathbb{G}_1$ and multi-pairings.

According to the post[1], an efficient non-conservative choice of a Groth16-tailored curve at the 128-bit security level is a BLS12 curve of roughly 384 bits. A conservative but efficient alternative is a BLS12 curve of 440 to 448 bits. Then to fulfill SNARK-friendliness, it is sufficient to choose a seed x s.t. $x \equiv 1 \mod 3 \cdot 2^L$ with the desired 2-adicity $L \geq 1$. Consequently, Prop. 3.1 and 4.1, and the faster co-factor clearing trick (cf. Section 3.2.1) apply: such an inner BLS12 is always of the form $Y^2 = X^3 + 1$; multiplying by $x-1$ is sufficient to clear the cofactor on $\mathbb{G}_1$, and the efficient $\mathbb{G}_T$ membership testing applies. In fact, for all BLS12 curves, $\gcd(p(x)+1-t(x), \Phi_{12}(p(x)))$ is always equal to $r(x)$ and the membership testing boils down to $z^p = (z^p)^p \cdot z$ and $z^p = z^u$ for $z \in \mathbb{G}_T$.

KZG-based SNARKs require a 128-bit secure curve with efficient $\mathbb{G}_1$ operations and fast pairing. For a faster $\mathbb{G}_1$ arithmetic, we consider a BLS24 curve of roughly 320 bits, that meets the 128-bit level security [GS21] and gives the best trade-off between small $\rho = \log_2 p / \log_2 r$ value ($\rho = 1.25$) and fast pairing. For SNARK-friendliness, cofactor

[1] `https://members.loria.fr/AGuillevic/pairing-friendly-curves/`

clearing and curve equation ($Y^2 = X^3 + 1$), the same observations as for BLS12 apply. For $\mathbb{G}_T$ membership testing, $\gcd(p(x)+1-t(x), \Phi_{24}(p(x)))$ is always equal to $r(x)$ for the BLS24 curves and the test boils down to $z^{p^2} = (z^{p^2})^{p^2} \cdot z$ and $z^p = z^u$ for $z \in \mathbb{G}_T$.

Remark 4.1. *For SNARK-friendly inner BLS, $z^u \in \mathbb{G}_T$ can be implemented efficiently by mixing Granger and Scott's [GS10] and Karabina's [Kar13] cyclotomic squares. Since $2^L \mid u-1$, there are $L-1$ consecutive squarings in the exponentiation. One can use Karabina's method for this series and then switch to Granger-Scott's method for the remaining part. Hence, trading off one inversion in $\mathbb{F}_{p^{k/d}}$ for $2(L-1)$ multiplications in $\mathbb{F}_{p^{k/d}}$. Particularly, for BLS12 and BLS24, this trick yields significant speedups as long as an inversion in $\mathbb{F}_p$ costs, respectively, less than $(6L-4)$ and $(18L-16)$ $\mathbb{F}_p$-multiplications, which is the case for the curves we are interested in. For instance, `gnark-crypto` implements an optimized x86-64 $\mathbb{F}_p$-multiplication [gna20] and Pornin's optimized GCD for $\mathbb{F}_p$-inversion [Por20]. For BLS12-377, the ratio of these operations is $\times 62$ and $L = 46$, and for BLS24-315 the ratio is $\times 60$ and $L = 20$ and 8 using custom short addition chains that maximize L's.*

4.4.2 Outer curves: Brezing–Weng, Cocks–Pinch

This section presents the families of 2-chains with a BW6 curve on top of an inner BLS12 curve, and on top of an inner BLS24 curve. Cocks-Pinch curves (CP) [FST10, §4.1] are addressed in Section 12. For BW6, all parameters and formulas are given as polynomials in the variable x, with integer parameters h_t, h_y that are the lifting cofactors of the Brezing–Weng construction. We use subscripts $p_{\text{bls}}, p_{\text{bw}}, p_\times$ to identify parameters of BLS, BW and CP curves. BW and CP constructions follow the same recipe, but CP deals with integers, while BW deals with polynomials [FST10, §4.1,§6]. They start from the subgroup order $r_{\text{bw}}(x) = p_{\text{bls}}(x)$, $r_\times(u) = p_{\text{bls}}(u)$, and look for k-th roots of unity $\zeta_k \bmod p_{\text{bls}}$ to set the trace value $t = \zeta_k + 1$. For CP, the existence of ζ_k requires $p_{\text{bls}}(u) \equiv 1 \bmod k$: for $k = 6, 12, 8$ resp., this means $u \equiv 1 \bmod 3$, $1, 10 \bmod 12$, and $1, 10 \bmod 24$ resp. For BW, the number field defined by $p_{\text{bls}}(x)$ only contains $\zeta_k(x)$ for $k \mid 6$, limiting the BW construction to $k = 6$ at most.

Generic BW6 curve parameters

To satisfy Definition 4.3, a BW curve chained to a BLS curve (of any embedding degree) has a subgroup of prime order $r_{\text{bw}}(x) = p_{\text{bls}}(x)$. To get an embedding degree $k = 6$, a primitive 6-th root of unity ζ_6 modulo $r_{\text{bw}}(x)$ is required, the trace of the curve modulo r_{bw} is then $t_{\text{bw},3} = \zeta_6 + 1 \bmod r_{\text{bw}}$. Alternatively $t_{\text{bw},0} = \overline{\zeta_6} + 1 \bmod r_{\text{bw}}$ with $\zeta_6 = -\overline{\zeta_6} + 1$. With $D = 3$ and $1/\sqrt{-3} = (2\overline{\zeta_6} - 1)/3 \bmod r_{\text{bw}}$, then $y_{\text{bw},0} = (t_{\text{bw},0} - 2)/\sqrt{-3} = (\overline{\zeta_6} + 1)/3 = -t_{\text{bw},0}/3$. Or with $1/\sqrt{-3} = -(2\zeta_6 - 1)/3 \bmod r_{\text{bw}}$, one has $y_{\text{bw},3} = (t_{\text{bw},3} - 2)/\sqrt{-3} = (\zeta_6 + 1)/3 = t_{\text{bw},3}/3$. Any BW6 curve will have parameters of the form $t_i = t_{\text{bw},i} \pm h_t r$, $y_i = y_{\text{bw},i} \pm h_y r$, where h_t, h_y are integer lifting cofactors. We label the two cases according to the constant coefficient of the polynomial defining the trace modulo r_{bw}: this is either 0 or 3.

One denotes $p_{\text{bw},0}(x, h_t, h_y) = ((t_{\text{bw},0} + h_t r)^2 + 3(y_{\text{bw},0} + h_y r)^2)/4$. We have

$$p_{\text{bw},0} = t_{\text{bw},0}^2/3 + t_{\text{bw},0} \cdot r_{\text{bw}}(h_t - h_y)/2 + r_{\text{bw}}^2(h_t^2 + 3h_y^2)/4 \,, \tag{4.2}$$

$$p_{\text{bw},3} = t_{\text{bw},3}^2/3 + t_{\text{bw},3} \cdot r_{\text{bw}}(h_t + h_y)/2 + r_{\text{bw}}^2(h_t^2 + 3h_y^2)/4 \,. \tag{4.3}$$

The curve cofactor $c_{\text{bw},i}(x, h_t, h_y)$ such that $c_{\text{bw},i} r_{\text{bw}} = p_{\text{bw},i} + 1 - t_{\text{bw},i}$ is

$$c_{\text{bw},0} = (h_t^2 + 3h_y^2)/4r_{\text{bw}} + (h_t - h_y)/2t_{\text{bw},0} + (t_{\text{bw},0}^2/3 - t_{\text{bw},0} + 1)/r_{\text{bw}} - h_t \tag{4.4}$$

$$c_{\text{bw},3} = (h_t^2 + 3h_y^2)/4r_{\text{bw}} + (h_t + h_y)/2t_{\text{bw},3} + (t_{\text{bw},3}^2/3 - t_{\text{bw},3} + 1)/r_{\text{bw}} - h_t \tag{4.5}$$

Table 4.5: Are $2(h_t \pm h_y)$, $h_t^2 + 3h_y^2$ multiple of 4?

h_t mod2	h_y mod2	$h_t \pm h_y$ mod2	$h_t^2 + 3h_y^2$ mod4	$2(h_t \pm h_y)t_{\mathrm{bw},i} + (h_t^2 + 3h_y^2)r_{\mathrm{bw}}$ mod 4	
				$t_{\mathrm{bw},0} = 0$ mod 2	$t_{\mathrm{bw},3} = 1$ mod 2
0	0	0	0	0	0
0	1	1	3	$3r_{\mathrm{bw}} \neq 0$	$2 + 3r_{\mathrm{bw}} \neq 0$
1	0	1	1	$r_{\mathrm{bw}} \neq 0$	$2 + r_{\mathrm{bw}} \neq 0$
1	1	0	0	0	0

where $(t_{\mathrm{bw},i}^2/3 - t_{\mathrm{bw},i} + 1)/r_{\mathrm{bw}} = \Phi_6(t_{\mathrm{bw},i} - 1)/(3r_{\mathrm{bw}})$ is a polynomial in $\mathbb{Q}[x]$ since by construction r_{bw} divides $\Phi_6(t_{\mathrm{bw},i} - 1)$. Tables 4.6 and 4.9 give the explicit values of the polynomials for BLS12 and BLS24 inner curves.

Cofactor of $\mathbb{G}_2$**.** The group $\mathbb{G}_2$ of order r_{bw} is a subgroup of one of the two sextic twists of E, defined over $\mathbb{F}_p$. Generically, the orders of the two sextic twists are $p+1-(t+3y)/2$ and $p+1-(t-3y)/2$, where y satisfies $t^2 - 4p = -3y^2$. One of the orders is a multiple of r_{bw}, and has cofactor $c'_{\mathrm{bw},i}$. Observe that $(t_{\mathrm{bw},0} - 3y_{\mathrm{bw},0})/2 = t_{\mathrm{bw},0}$ since $y_{\mathrm{bw},0} = -t_{\mathrm{bw},0}/3$. The correct sextic twist has order

$$\begin{aligned}
&p_{\mathrm{bw},0} + 1 - (t_{\mathrm{bw},0} + h_t r_{\mathrm{bw}} - 3(y_{\mathrm{bw},0} + h_y r_{\mathrm{bw}}))/2 \\
&= p_{\mathrm{bw},0} + 1 - \underbrace{(t_{\mathrm{bw},0} - 3y_{\mathrm{bw},0})/2}_{=t_{\mathrm{bw},0}} - h_t r_{\mathrm{bw}}/2 + 3h_y r_{\mathrm{bw}}/2 \\
&= \underbrace{p_{\mathrm{bw},0} + 1 - t_{\mathrm{bw},0} - h_t r_{\mathrm{bw}}}_{=\#E(\mathbb{F}_p) = r_{\mathrm{bw}} \cdot c_{\mathrm{bw},0}} + (h_t r_{\mathrm{bw}} + 3h_y r_{\mathrm{bw}})/2 \\
&= r_{\mathrm{bw}} \cdot \underbrace{(c_{\mathrm{bw},0} + (h_t + 3h_y)/2)}_{c'_{\mathrm{bw},0}}
\end{aligned}$$

hence

$$c'_{\mathrm{bw},0} = c_{\mathrm{bw},0} + (h_t + 3h_y)/2 \ . \tag{4.6}$$

For the other trace, $(t_{\mathrm{bw},3} + 3y_{\mathrm{bw},3})/2 = t_{\mathrm{bw},3}$ and the correct sextic twist has order $p_{\mathrm{bw},3} + 1 - (t_{\mathrm{bw},3} + h_t r_{\mathrm{bw}} + 3(y_{\mathrm{bw},3} + h_y r_{\mathrm{bw}}))/2$ a multiple of r_{bw}, and cofactor

$$c'_{\mathrm{bw},3} = c_{\mathrm{bw},3} + (h_t - 3h_y)/2 \ . \tag{4.7}$$

Congruences of cofactors h_t***,*** h_y***.*** One requires $p_{\mathrm{bw},i}$ (Eqs. (4.2), (4.3)) to be an integer and a prime. Because $t_{\mathrm{bw},i}$ is always multiple of 3, $t_{\mathrm{bw},i}^2/3$ is an integer. We need $(h_t \pm h_y)/2t_{\mathrm{bw},i} + (h_t^2 + 3h_y^2)/4r_{\mathrm{bw}}$ to be an integer. We now look at $(h_t \pm h_y)$, $(h_t^2 + 3h_y^2)$. We have $t_{\mathrm{bw},0}$ always even, then $(h_t - h_y)t_{\mathrm{bw},0}/2$ is an integer and we require $4 \mid (h_t^2 + 3h_y^2)$. For that we need $h_t - h_y \equiv 0 \bmod 2$ (see Table 4.5). We have $t_{\mathrm{bw},3}$ always odd. If $(h_t + h_y)$ is odd, then $(h_t + h_y)t_{\mathrm{bw},3}$ is odd but at the same time (see Table 4.5), $(h_t^2 + 3h_y^2)$ is odd, and the condition is not satisfied. Hence we need $(h_t - h_y)$ to be even, and consequently we have $(h_t^2 + 3h_y^2)/4$ an integer. Finally, for both $t_{\mathrm{bw},0}$ and $t_{\mathrm{bw},3}$, we need $2 \mid (h_t - h_y)$ and consequently we have $4 \mid h_t^2 + 3h_y^2$, to ensure p_{bw} to be an integer. Note also that because $x \equiv 1 \bmod 3$, one has $t_{\mathrm{bw}} = 0 \bmod 3$, and Eqs. (4.2), (4.3) give $4p_{\mathrm{bw}} = h_t^2 \bmod 3$. Because p_{bw} needs to be prime, h_t is not multiple of 3, and $3 \nmid (h_t^2 + 3h_y^2)$.

Subgroup membership testing: $\mathbb{G}_T$**.** We apply the technique of Section 3.2.2. BW6 curves over their base field have order $c_{\mathrm{bw},i} \cdot r_{\mathrm{bw}} = p_{\mathrm{bw},i} + 1 - t_{\mathrm{bw},i} - h_t r_{\mathrm{bw}}$, hence

$$p_{\mathrm{bw},i} \equiv t_{\mathrm{bw},i} - 1 \bmod r_{\mathrm{bw}} \ . \tag{4.8}$$

As soon as $\gcd(p_{\mathrm{bw},i} + 1 - t_{\mathrm{bw},i}, \Phi_k(p_{\mathrm{bw},i})) = r_{\mathrm{bw}}$, then the following two tests are enough:

1. test if $z^{\Phi_k(p_{\mathrm{bw},i})} = 1$ with Frobenius maps;
2. test if $z^{p_{\mathrm{bw},i}} = z^{t_{\mathrm{bw},i}-1}$ with cyclotomic squarings.

Easy part of the final exponentiation. the final exponentiation raises the Miller loop output f to the power

$$(p^6 - 1)/r = (p^6 - 1)/\Phi_6(p) \cdot \Phi_6(p)/r = (p^3 - 1)(p + 1)(p^2 - p + 1)/r \ .$$

The easy part $(p^3 - 1)(p + 1)$ costs one conjugation (p^3-Frobenius power), one inversion in $\mathbb{F}_{p^6}$, one p-Frobenius power and two multiplications. We optimise the hard part $(p^2-p+1)/r$ in Section 21, 14.

Optimal Pairing Computation. In [EHG20], we presented an optimal ate pairing formula that can be generalized as follows: write

$$a_0 + a_1(t_{\mathrm{bw},i} - 1) = 0 \bmod r_{\mathrm{bw}} \tag{4.9}$$

with shortest possible scalars a_0, a_1. On $\mathbb{G}_2$, the Frobenius π_p has eigenvalue $t_{\mathrm{bw},i} - 1$. The optimal ate Miller loop is computed with the formula

$$f_{a_0,Q}(P) f_{a_1,\pi_p(Q)}(P) = f_{a_0,Q}(P) f^p_{a_1,Q}(P) \ . \tag{4.10}$$

Moreover, it turned out that $(a_1 - 1) \mid a_2$, and some of the computations were shared. We now introduce another optimisation. We consider Eq. (4.9) with a new point of view. BW6 curves have an endomorphism $\phi\colon (x, y) \mapsto (\omega x, -y)$ on $\mathbb{G}_1$ of eigenvalue $\lambda = t_{\mathrm{bw},i} - 1 = p_{\mathrm{bw},i} \bmod r_{\mathrm{bw}}$, and characteristic polynomial $\chi^2 - \chi + 1 = 0$. The (bilinear) twisted ate pairing [HSV06, §6] has precisely Miller loop $f_{\lambda,P}(Q)$. However, λ is too large for a fast pairing computation so instead, we consider a multiple of the Tate pairing $f_{hr,P}(Q) = f_{a_0+a_1\lambda,P}(Q)$ for some h (e.g. Eqs.(4.19), (4.26)). Instead of decomposing the Miller function $f_{a_0+a_1\lambda,P}(Q)$ into sub-functions $f_{a_0,P}(Q) f_{a_1\lambda,P}(Q)$, we use Lemma 4.5 to get shared squares in $\mathbb{F}_{p^k}$ and shared doubling steps in $\mathbb{G}_1$ (Tate), resp. $\mathbb{G}_2$ (ate), in the same idea as a multi-scalar multiplication. This gives us Alg. 4.2. We are in the very particular case of $k/d = 1$, ϕ on $\mathbb{G}_1$ and π_p on $\mathbb{G}_2$ both have eigenvalue $p_{\mathrm{bw},i} \bmod r_{\mathrm{bw}}$, and our variant of the twisted ate pairing is competitive with the ate pairing.

Lemma 4.5. *Let E be a pairing-friendly curve with the usual order-r subgroups $\mathbb{G}_1, \mathbb{G}_2$, two points $P \in \mathbb{G}_i$, $Q \in \mathbb{G}_{1-i}$ of order r, and an endomorphism ϕ of eigenvalue λ over $\mathbb{G}_i$: $\phi(P) = [\lambda]P$, $\lambda = p^e \bmod r$ for some $1 \le e \le k - 1$. The Miller function can be decomposed as follows.*

$$\begin{aligned}
f_{2(u+v\lambda),P}(Q) &= f^2_{u+v\lambda,P}(Q) \ell_{(u+v\lambda)P,(u+v\lambda)P}(Q) && (4.11)\\
f_{u+1+v\lambda,P}(Q) &= f_{u+v\lambda,P}(Q) \ell_{(u+v\lambda)P,P}(Q) && (4.12)\\
f_{u+(v+1)\lambda,P}(Q) &= f_{u+v\lambda,P}(Q) \ell_{(u+v\lambda)P,\lambda P}(Q) && (4.13)\\
f_{u+1+(v+1)\lambda,P}(Q) &= f_{u+v\lambda,P}(Q) \ell_{P,\lambda P}(Q) \ell_{(u+v\lambda)P,(1+\lambda)P}(Q) && (4.14)
\end{aligned}$$

where $\lambda P = \phi(P)$, $(1 + \lambda)P = P + \phi(P)$, and $\ell_{P,\lambda P}(Q)$ can be precomputed.

Algorithm 4.2: Miller loop for optimal pairing with endomorphism ϕ on $\mathbb{G}_1$ (Tate), resp. $\mathbb{G}_2$ (ate) of eigenvalue λ and degree 2.
Input: $P \in \mathbb{G}_i$, $Q \in \mathbb{G}_{1-i}$, end. ϕ on $\mathbb{G}_i$ of eigenvalue λ, scalars a_0, a_1 s. t. $a_0 + a_1\lambda = 0 \bmod r$
Output: $f_{a_0+a_1\lambda,P}(Q)$

$P_0 \leftarrow P$; $P_1 \leftarrow \phi(P)$
if $a_0 < 0$**:** $a_0 \leftarrow -a_0$; $P_0 \leftarrow -P_0$
if $a_1 < 0$**:** $a_1 \leftarrow -a_1$; $P_1 \leftarrow -P_1$
$P_{1+\lambda} \leftarrow P_0 + P_1$; $\ell_{1,\lambda} \leftarrow \ell_{P_0,P_1}(Q)$
$l_0 \leftarrow \text{bits}(a_0)$; $l_1 \leftarrow \text{bits}(a_1)$
if $\#l_0 = \#l_1$**:** $S \leftarrow P_{1+\lambda}$; $f \leftarrow \ell_{1,\lambda}$; $n \leftarrow \#l_0$
elif $\#l_0 < \#l_1$**:** $S \leftarrow P_1$; $f \leftarrow 1$; $n \leftarrow \#l_1$; pad l_0 with 0 s.t. $\#l_0 = n$
else: $S \leftarrow P_0$; $f \leftarrow 1$; $n \leftarrow \#l_0$; pad l_1 with 0 s.t. $\#l_1 = n$
for $i = n-2$ *downto* 0**:**
    $f \leftarrow f^2$; $\ell_t \leftarrow \ell_{S,S}(Q)$; $S \leftarrow [2]S$
    if $l_0[i] = 0$ *and* $l_1[i] = 0$**:** $f \leftarrow f \cdot \ell_t$ // Eq. (4.11), $m_{\text{full-sparse}}$
    elif $l_0[i] = 1$ *and* $l_1[i] = 1$**:** // Eq. (4.14)
        $S \leftarrow S + P_{1+\lambda}$; $\ell \leftarrow \ell_{S,P_{1+\lambda}}(Q)$
        $f \leftarrow (f \cdot \ell_t) \cdot (\ell \cdot \ell_{1,\lambda})$ // $m_k + m_{\text{full-sparse}} + m_{\text{sparse-sparse}}$
    elif $l_0[i] = 1$**:** // Eq. (4.12)
        $S \leftarrow S + P_0$; $\ell \leftarrow \ell_{S,P_0}(Q)$
        $f \leftarrow f \cdot (\ell_t \cdot \ell)$ // $m_k + m_{\text{sparse-sparse}}$
    else: $(l_1[i] = 1)$ // Eq. (4.13)
        $S \leftarrow S + P_1$; $\ell \leftarrow \ell_{S,P_1}(Q)$
        $f \leftarrow f \cdot (\ell_t \cdot \ell)$ // $m_k + m_{\text{sparse-sparse}}$
return f

Proof (of Lemma 4.5). The usual Miller formulas give (see e.g. [Ver10])

$$\begin{aligned}
f_{2(u+v\lambda),P}(Q) &= f^2_{u+v\lambda,P}(Q)\ \underbrace{f_{2,[u+v\lambda]P}(Q)}_{=\text{ tangent at } (u+v\lambda)P} \\
f_{u+1+v\lambda,P}(Q) &= f_{u+v\lambda,P}(Q)\ \underbrace{f_{1,P}(Q)}_{=1}\ \ell_{(u+v\lambda)P,P}(Q) \\
f_{u+(v+1)\lambda,P}(Q) &= f_{u+v\lambda,P}(Q)\ \underbrace{f_{\lambda,P}(Q)}_{\text{bilinear pairing}}\ \ell_{(u+v\lambda)P,\lambda P}(Q) \\
f_{u+1+(v+1)\lambda,P}(Q) &= f_{u+v\lambda,P}(Q)\ \underbrace{f_{1+\lambda,P}(Q)}_{f_{1,P}(Q)f_{\lambda,P}(Q)\ell_{P,\lambda P}(Q)}\ \ell_{(u+v\lambda)P,(1+\lambda)P}(Q)
\end{aligned}$$

The term $f_{1,P}(Q) = 1$ can disappear. The term $f_{\lambda,P}(Q)$ is a bilinear pairing as $\lambda \equiv p^e \bmod r$, and then can be removed. Finally $f_{1+\lambda,P}(Q)$ simplifies to $\ell_{P,\lambda P}(Q)$ which can be precomputed. □

Remark 4.2. *Alg. 4.2 shares the squarings in $\mathbb{F}_{p^k}$ and the doubling steps in $\mathbb{G}_1$ (Tate), resp. $\mathbb{G}_2$ (ate). With all parameterized pairing-friendly families, the scalar decomposition gives all but one trivial Miller function, and the ate, or twisted-ate pairing boils down to one Miller loop computation of optimal length, and a few line additions [Ver10]. In*

Table 4.6: Parameters of a BW6 outer curve with a BLS12 inner curve, with $x \equiv 1 \bmod 3$.

parameter	value	property
r_{bw}	$p_{\text{bls}} = (x-1)^2/3(x^4-x^2+1)+x$	generates prime
$\overline{\zeta_6}$	$-x^5+3x^4-3x^3+x-1$	
ζ_6	$x^5-3x^4+3x^3-x+2$	
$1/\sqrt{-3}$	$-(2x^5-6x^4+6x^3-2x+3)/3$	
$t_{\text{bw},0}$	$-x^5+3x^4-3x^3+x$	$6 \mid t_{\text{bw},0}$
$t_{\text{bw},3}$	$x^5-3x^4+3x^3-x+3$	$3 \mid t_{\text{bw},3}, 2 \nmid t_{\text{bw},3}$
$y_{\text{bw},0}$	$(x^5-3x^4+3x^3-x)/3 = -t_{\text{bw},0}/3$	$2 \mid y_{\text{bw},0}$
$y_{\text{bw},3}$	$(x^5-3x^4+3x^3-x+3)/3 = t_{\text{bw},3}/3$	$2 \nmid y_{\text{bw},3}$
$p_{\text{bw},0}$	$((t_{\text{bw},0}+h_t r_{\text{bw}})^2+3(y_{\text{bw},0}+h_y r_{\text{bw}})^2)/4$	generates prime
$p_{\text{bw},3}$	$((t_{\text{bw},3}+h_t r_{\text{bw}})^2+3(y_{\text{bw},3}+h_y r_{\text{bw}})^2)/4$	generates prime
$\Phi_6(t_{\text{bw},i}-1)$	$3r_{\text{bw}}(x^4-4x^3+7x^2-6x+3)$	
$c_{\text{bw},0}$	$(h_t^2+3h_y^2)/4r_{\text{bw}}+(h_t-h_y)/2t_{\text{bw},0}+x^4-4x^3+7x^2-6x+3-h_t$	
$c_{\text{bw},3}$	$(h_t^2+3h_y^2)/4r_{\text{bw}}+(h_t+h_y)/2t_{\text{bw},3}+x^4-4x^3+7x^2-6x+3-h_t$	
$c'_{\text{bw},0}$ ($\mathbb{G}_2$)	$c_{\text{bw},0}+(h_t+3h_y)/2$	
$c'_{\text{bw},3}$ ($\mathbb{G}_2$)	$c_{\text{bw},3}+(h_t-3h_y)/2$	
Optimal ate Miller loop $f_{a_0,Q}(P)\cdot f^p_{a_1,Q}(P)$, Optimal twisted ate $f_{a_0+a_1\lambda,P}(Q)$		
$\lambda = t_{\text{bw},0}-1$	$a_0 = x^3-x^2-x,\ a_1 = x+1 \quad a'_0 = -(x+1),\ a'_1 = x^3-x^2+1$	
$\lambda = t_{\text{bw},3}-1$	$a_0 = x+1,\ a_1 = x^3-x^2-x \quad a'_0 = x^3-x^2+1,\ a'_1 = -(x+1)$	
Final exponentiation		
$e_{\text{bw},0}$	$(x+1)\Phi_6(p_{\text{bw},0})/r_{\text{bw}} =$ $((x+1)p_{\text{bw},0}-x^3+x^2-1)(c_{\text{bw},0}+h_t)+3(p_{\text{bw},0}-x^2+2x-2)$	
$e'_{\text{bw},0}$	$(x^3-x^2-x)\Phi_6(p_{\text{bw},0})/r_{\text{bw}} =$ $((x^3-x^2-x)p_{\text{bw},0}+x+1)(c_{\text{bw},0}+h_t)-3(1+(x^2-2x+1)p_{\text{bw},0})$	
$e_{\text{bw},3}$	$(x+1)\Phi_6(p_{\text{bw},3})/r_{\text{bw}} =$ $(x^3-x^2-x+(x+1)p_{\text{bw},3})(c_{\text{bw},3}+h_t)+3(x^2-2x+1+p_{\text{bw},3})$	
$e'_{\text{bw},3}$	$(x^3-x^2+1)\Phi_6(p_{\text{bw},3})/r_{\text{bw}} =$ $((x^3-x^2+1)p_{\text{bw},3}-x-1)(c_{\text{bw},3}+h_t)-3(1-(x^2-2x+2)p_{\text{bw},3})$	

our case, while being short, none of the scalars a_0, a_1 is trivial. It is possible to derive a 2-NAF variant of Alg. 4.2 (see Table 4.14). It requires the additional precomputations of $P-\phi(P)$ and $\ell_{P,-\lambda P}(Q)$. From the estimate in Table 4.14, our Miller loop variant in Alg. 4.2 would give up to a 7% speed-up compared to [EHG20, Alg. 5], for BLS24-BW6 curves. Our Alg. 4.2 works for Tate and ate pairing. If there is an endomorphism of higher degree on $\mathbb{G}_2$ (or two independent endomorphisms), use Alg. 4.10 instead.

BW6 with BLS-12

Table 4.6 gives the parameters of the BW6-BLS12 curves in terms of the seed x, and the two lifting cofactors h_t, h_y.

Optimal Ate Pairing Computation. We investigate two pairings on our BW6 curves: optimal ate and optimal Tate. In [EHG20], we presented an optimal ate pairing formula, for any BW6 curve with $t_{\text{bw},3}$:

$$m_{\text{opt. ate}} = f_{u+1,Q}(P) f^p_{u^3-u^2-u,Q}(P) \quad \text{and } e_{\text{opt. ate}} = m_{\text{opt. ate}}^{(p_{\text{bw}}^6-1)/r_{\text{bw}}} \tag{4.15}$$

with optimized computation in [EHG20, Alg. 5]:

$$f_u = f_{u,Q}(P);\ m_{\text{opt. ate}} = f_u \cdot (f_u)^p_{u^2-u-1,[u]Q}(P)\ell_{[u]Q,Q}(P)\ , \tag{4.16}$$

where $[u]Q$ is a side result of the computation $f_{u,Q}(P)$. The equivalent formula for a trace $t_{\text{bw},0}$ is

$$f_{u(u^2-u-1),Q}(P)f^p_{u+1,Q}(P) \tag{4.17}$$

whose optimized version is

$$f_u = f_{u,Q}(P);\ m_{\text{opt. ate}} = (f_u \cdot \ell_{[u]Q,Q}(P))^p(f_u)_{u^2-u-1,[u]Q}(P)\ . \tag{4.18}$$

In the two cases $t_{\text{bw},0}$ and $t_{\text{bw},3}$, the cost in terms of multiplications in the base field are the same.

Optimal Pairing Computation with Alg. 4.2. $\mathbb{G}_1$ and $\mathbb{G}_2$ have an endomorphism ϕ_1, ϕ_2 of eigenvalue $\lambda_{\text{bw},i} = t_{\text{bw},i} - 1 \bmod r_{\text{bw}}$. Low degree polynomials (short scalars once evaluated at a seed u) a_0, a_1 s.t. $a_0 + a_1\lambda_{\text{bw},i} = 0 \bmod r_{\text{bw}}$ are

$$\begin{aligned}
(x^3 - x^2 - x) + (x+1)(t_{\text{bw},0} - 1) &= -3r_{\text{bw}} && (4.19)\\
-x - 1 + (x^3 - x^2 + 1)(t_{\text{bw},0} - 1) &= -3(x^2 - 2x + 2)r_{\text{bw}} && (4.20)\\
(x+1) + (x^3 - x^2 - x)(t_{\text{bw},3} - 1) &= 3(x-1)^2 r_{\text{bw}} && (4.21)\\
(x^3 - x^2 + 1) - (x+1)(t_{\text{bw},3} - 1) &= -3r_{\text{bw}} && (4.22)
\end{aligned}$$

The optimal Tate or ate Miller loop with e.g. (4.20), (4.22) are:

$$\begin{aligned}
m_{\text{Tate}} = f_{-u-1+(u^3-u^2+1)\lambda_{\text{bw},0},P}(Q),\ m_{\text{ate}} = f_{-u-1+(u^3-u^2+1)p_{\text{bw},0},Q}(P)\\
m_{\text{Tate}} = f_{u^3-u^2+1-(u+1)\lambda_{\text{bw},3},P}(Q),\ m_{\text{ate}} = f_{u^3-u^2+1-(u+1)p_{\text{bw},3},Q}(P)\ .
\end{aligned}$$

$\mathbb{G}_1$ and $\mathbb{G}_2$ membership testing. For $\mathbb{G}_1$ membership testing, one uses one of Eqs. (4.19), (4.20), resp. (4.21), (4.22), with $x = u$. However, these formulas (e.g. $[u^3 - u^2 - u]P + [u+1]\phi(P)$) will output $\mathcal{O}$ for any point in the subgroup of order $3r_{\text{bw}}$. For $\mathbb{G}_2$ membership testing, the same equations can be re-used: we showed in Section 4.4.2 that the twisted curve E' of $\mathbb{G}_2$ has the same trace as E modulo r_{bw}, either $(t_{\text{bw},0} - 3y_{\text{bw},0})/2 = t_{\text{bw},0}$, or $(t_{\text{bw},3} + 3y_{\text{bw},3})/2 = t_{\text{bw},3}$.

Final Exponentiation. Writing the hard part of the final exponentiation $z^{\Phi_6(p_{\text{bw},i})/r_{\text{bw}}}$ in terms of x, h_t, h_y, Magma runs Lenstra–Lenstra–Lovász (LLL) lattice basis reduction algorithm on multivariate polynomials and provides the result. With $t_{\text{bw},i}$, LLL gives short vectors for the exponent:

$$e_{\text{bw},i} = 3(x+1)\Phi_k(p_{\text{bw},i})/r_{\text{bw}}(x) \tag{4.23}$$

and the formulas for $e_{\text{bw},i}$ are

$$\begin{aligned}
e_{\text{bw},0} &= 3(c_{\text{bw},0}+h_t)(-x^3+x^2-1+(x+1)p_{\text{bw},0}) + 9(x^2-2x+2-p_{\text{bw},0}) && (4.24)\\
e_{\text{bw},3} &= 3(c_{\text{bw},3}+h_t)(x^3-x^2-x+(x+1)p_{\text{bw},3}) + 9(x^2-2x+1+p_{\text{bw},3}) && (4.25)
\end{aligned}$$

Alternatively we obtain

$$\begin{aligned}
e'_{\text{bw},0} &= 3(x^3 - x^2 - x)\Phi_k(p_{\text{bw},0})/r_{\text{bw}}(x)\\
e'_{\text{bw},3} &= 3(x^3 - x^2 + 1)\Phi_k(p_{\text{bw},3})/r_{\text{bw}}(x)
\end{aligned}$$

Curve	Cost in terms of operations in $\mathbb{F}_{p^6}$
BN-BW6, t_0, Alg. 4.7	$6\mathbf{e}^{x} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 6\mathbf{s} + 14\mathbf{m} + \mathbf{f} + 3\mathbf{cj}$
BN-BW6, t_0, alternative	$6\mathbf{e}^{x} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 6\mathbf{s} + 14\mathbf{m} + \mathbf{f} + 4\mathbf{cj}$
BN-BW6, t_3, alternative	$6\mathbf{e}^{x} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 6\mathbf{s} + 15\mathbf{m} + \mathbf{f} + 2\mathbf{cj}$
BN-BW6, t_3, Alg. 4.8	$6\mathbf{e}^{x} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 6\mathbf{s} + 15\mathbf{m} + \mathbf{f}$
BLS12-BW6, t_0, Alg. 4.3	$5\mathbf{e}^{x-1} + \mathbf{e}^{(x-1)/3} + 3\mathbf{e}^{x+1} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 14\mathbf{m} + 2\mathbf{s} + \mathbf{f} + 4\mathbf{cj}$
BLS12-BW6, t_0, alt.	$5\mathbf{e}^{x-1} + \mathbf{e}^{(x-1)/3} + 3\mathbf{e}^{x+1} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 15\mathbf{m} + 2\mathbf{s} + \mathbf{f} + 4\mathbf{cj}$
BLS12-BW6, t_3, Alg. 4.4	$5\mathbf{e}^{x-1} + \mathbf{e}^{(x-1)/3} + 3\mathbf{e}^{x+1} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 14\mathbf{m} + 2\mathbf{s} + \mathbf{f} + 3\mathbf{cj}$
BLS12-BW6, t_3, alt.	$5\mathbf{e}^{x-1} + \mathbf{e}^{(x-1)/3} + 3\mathbf{e}^{x+1} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 15\mathbf{m} + 2\mathbf{s} + \mathbf{f} + 4\mathbf{cj}$
BLS24-BW6, t_0, Alg. 4.5	$5\mathbf{e}^{x-1} + \mathbf{e}^{(x-1)/3} + 3\mathbf{e}^{x^2+1} + 3\mathbf{e}^{x+1} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 14\mathbf{m} + 2\mathbf{s} + \mathbf{f} + 4\mathbf{cj}$
BLS24-BW6, t_0, alt.	$5\mathbf{e}^{x-1} + \mathbf{e}^{(x-1)/3} + 3\mathbf{e}^{x^2+1} + 3\mathbf{e}^{x+1} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 15\mathbf{m} + 2\mathbf{s} + \mathbf{f} + 4\mathbf{cj}$
BLS24-BW6, t_3, Alg. 4.6	$5\mathbf{e}^{x-1} + \mathbf{e}^{(x-1)/3} + 3\mathbf{e}^{x^2+1} + 3\mathbf{e}^{x+1} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 15\mathbf{m} + 2\mathbf{s} + \mathbf{f} + 2\mathbf{cj}$
BLS24-BW6, t_3, alt.	$5\mathbf{e}^{x-1} + \mathbf{e}^{(x-1)/3} + 3\mathbf{e}^{x^2+1} + 3\mathbf{e}^{x+1} + \mathbf{e}^{h_2} + \mathbf{e}^{h_1} + 16\mathbf{m} + 2\mathbf{s} + \mathbf{f} + 3\mathbf{cj}$

Table 4.7: Cost of hard part of final exponentiation. $\mathbf{e}^{y}$ stands for exponentiation to y, $\mathbf{m}$ is multiplication, $\mathbf{s}$ is squaring, $\mathbf{f}$ is Frobenius (x^p), $\mathbf{cj}$ is conjugation, all in $\mathbb{F}_{p^6}$. For BLS24-BW6 curves, $m^{x^2+1} = (m^x)^x \cdot m$. If x^2+1 has a lower Hamming weight than $2\,\mathrm{Hw}(x)+1$, it is faster to do the former.

and the formulas for $e'_{\mathrm{bw},i}$, where $e'_{\mathrm{bw},3}$ matches with the formulas in [EHG20], are

$$e'_{\mathrm{bw},0} = 3(c_{\mathrm{bw},0}+h_t)((x^3-x^2-x)p_{\mathrm{bw},0}+x+1) - 9(1+(x^2-2x+1)p_{\mathrm{bw},0})$$
$$e'_{\mathrm{bw},3} = 3(c_{\mathrm{bw},3}+h_t)((x^3-x^2+1)p_{\mathrm{bw},3}-x-1) - 9(1-(x^2-2x+2)p_{\mathrm{bw},3})$$

Later in [AEHG22], inspired by the work of Hayashida, Hayasaka and Teruya [HHT20] and the work of Cai, Hu and Zhao [CHZ22], we improve the hard part computation. The key-ingredient is to factor as many terms as possible. The cost summary is in Tab. 4.7. The SageMath implementation is at `https://gitlab.inria.fr/zk-curves/snark-2-chains`. Parameters are in Tab. 4.6, where $d = \Phi_6(t_i-1)/(3r) = (x^4-4x^3+7x^2-6x+3)$. We highlight with an *underbrace* the same exponent a_i that can be shared. The steps in Alg. 4.3 correspond to the exponents

$$\begin{array}{rclcrcl}
a & = & (x-1)/3 & & & & \\
b & = & a(x-1) & & & = & (x-1)^2/3 \\
c & = & b((x-1)^2+1) & = & (d-1)/3 & = & (x-1)^2/3(x^2-2x+2) \\
e & = & -(u+1)c+b-a & = & t_0/3 & = & (x-1)^2/3(-x^3+x^2-1)-(x-1)/3 \\
f & = & -(u+1)(e+b)+a+1 & = & r & = & (x-1)^2/3(x^4-x^2+1)+x
\end{array}$$

The steps in Alg. 4.4 correspond to this sequence deduced from the former, with $t_3/3 = -t_0+1$.

$$\begin{array}{rclcl}
a & = & (x-1)/3 & & \\
b & = & a(x-1) & = & (x-1)^2/3 \\
c & = & b((x-1)^2+1) & = & (d-1)/3 \\
b' & = & -b & & \\
e & = & b'+1 & & \\
f & = & c(x+1)+e & = & (x-1)^2/3(x^3-x^2+1)+1 \\
g & = & f+a & = & t_3/3 \\
h & = & (f+b')(x+1)-e & = & r
\end{array}$$

$$(x+1)\frac{\Phi_6(p_0)}{r} = \overbrace{\big((x+1)\underbrace{(p-(x-1)^2-1)}_{a_0}+1\big)}^{b_0}\Big(\tfrac{h_t^2+3h_y^2}{4}r+3\big(\tfrac{h_t-h_y}{2}\tfrac{t_0}{3}+\tfrac{d-1}{3}\big)+1\Big)-3\underbrace{(p-(x-1)^2-1)}_{a_0}$$

$$(x^3-x^2-x)\frac{\Phi_6(p_0)}{r} = \overbrace{\big((x+1)\underbrace{((x-1)^2p+1)}_{a'_0}-p\big)}^{b'_0}\Big(\tfrac{h_t^2+3h_y^2}{4}r+3\big(\tfrac{h_t-h_y}{2}\tfrac{t_0}{3}+\tfrac{d-1}{3}\big)+1\Big)-3\underbrace{((x-1)^2p+1)}_{a'_0}$$

$$(x+1)\frac{\Phi_6(p_3)}{r} = \overbrace{\big((x+1)\underbrace{((x-1)^2+p)}_{a_3}-1\big)}^{b_3}\Big(\tfrac{h_t^2+3h_y^2}{4}r+3\big(\tfrac{h_t+h_y}{2}\tfrac{t_3}{3}+\tfrac{d-1}{3}\big)+1\Big)+3\underbrace{((x-1)^2+p)}_{a_3}$$

$$(x^3-x^2+1)\frac{\Phi_6(p_3)}{r} = \overbrace{\big((x+1)\underbrace{(((x-1)^2+1)p-1)}_{a'_3}-p\big)}^{b'_3}\Big(\tfrac{h_t^2+3h_y^2}{4}r+3\big(\tfrac{h_t+h_y}{2}\tfrac{t_3}{3}+\tfrac{d-1}{3}\big)+1\Big)+3\underbrace{(((x-1)^2+1)p-1)}_{a'_3}$$

Algorithm 4.3: Hard part of final exp., BLS12-BW6, t_0
Input: x, m, $h_1=(h_t-h_y)/2$, $h_2=(h_t^2+3h_y^2)/4$
Output: $m^{(x^3-x^2-x)\Phi_6(p_0)/r}$

1 $Q \leftarrow m^p$
2 $A \leftarrow (Q^{x-1})^{x-1}\cdot m$ ▷$m^{a'_0}$
3 $B \leftarrow A^{x+1}\cdot\overline{Q}$ ▷$m^{b'_0}$
4 $A \leftarrow \overline{A^2\cdot A}$ ▷$m^{-3a'_0}$
5 $C \leftarrow B^{(x-1)/3}$
6 $D \leftarrow C^{x-1}$
7 $E \leftarrow (D^{x-1})^{x-1}\cdot D$ ▷$B^{(d-1)/3}$
8 $F \leftarrow \overline{E^{x+1}\cdot C}\cdot D$ ▷$B^{t_0/3}$
9 $G \leftarrow \overline{(F\cdot D)^{x+1}}\cdot C\cdot B$ ▷B^r
10 $H \leftarrow F^{h_1}\cdot E$
11 $H \leftarrow H^2\cdot H\cdot B\cdot G^{h_2}$
12 **return** $A\cdot H$

Algorithm 4.4: Hard part of final exp., BLS12-BW6, t_3
Input: x, m, $h_1=(h_t+h_y)/2$, $h_2=(h_t^2+3h_y^2)/4$
Output: $m^{(x+1)\Phi_6(p_3)/r}$

1 $A \leftarrow m^p\cdot(A^{x-1})^{x-1}$ ▷m^{a_3}
2 $B \leftarrow \overline{m}\cdot A^{x+1}$ ▷m^{b_3}
3 $A \leftarrow A^2\cdot A$ ▷m^{3a_3}
4 $C \leftarrow B^{(x-1)/3}$
5 $D \leftarrow C^{x-1}$
6 $E \leftarrow (D^{x-1})^{x-1}\cdot D$ ▷$B^{(d-1)/3}$
7 $D \leftarrow \overline{D}$
8 $F \leftarrow D\cdot B$
9 $G \leftarrow E^{x+1}\cdot F$
10 $H \leftarrow G\cdot C$ ▷$B^{t_3/3}$
11 $I \leftarrow (G\cdot D)^{x+1}\cdot\overline{F}$ ▷B^r
12 $J \leftarrow H^{h_1}\cdot E$
13 $K \leftarrow J^2\cdot J\cdot B\cdot I^{h_2}$
14 **return** $A\cdot K$

Cofactor clearing on $\mathbb{G}_1$ and $\mathbb{G}_2$ with one endomorphism. The cofactors are $c_{\mathrm{bw},i}$ for $\mathbb{G}_1$ (Eqs. (4.4), (4.5)), resp. $c'_{\mathrm{bw},i}$ for $\mathbb{G}_2$ (Eqs. (4.6), (4.7)), Table 4.6. The curve and its sextic twist for $\mathbb{G}_2$ have an endomorphism defined over $\mathbb{F}_p$, of characteristic polynomial x^2+x+1 and eigenvalue λ such that $\lambda^2+\lambda+1=0$ modulo the curve order. There are two formulas, one for each choice of eigenvalue $\lambda, \overline{\lambda}=-\lambda-1$ modulo the curve order, and $l_0+l_1\lambda=0 \bmod c_{\mathrm{bw},i}$, resp. modulo $c'_{\mathrm{bw},i}$, resp. $\overline{l_0}+\overline{l_1}\lambda=0 \bmod c_{\mathrm{bw},i}$, resp. modulo $c'_{\mathrm{bw},i}$, summarized in Table 4.8. The formulas are implemented in SageMath, GIT at [EHG21].

BW6 with BLS-24

We follow the same process as for BW6-BLS12 and report the parameters in Table 4.9.

Table 4.8: Cofactor clearing on $\mathbb{G}_i$ with an endomorphism of eigenvalue $\lambda, \overline{\lambda}$.

$\mathbb{G}_1$	$c_{\mathrm{bw},0}$	$l_0 = (h_t^2+3h_y^2)/4\cdot(x^3-x^2+1) - h_t(x^2-2x+1) - (h_t-3h_y)/2$
		$l_1 = (h_t^2+3h_y^2)/4\cdot(x+1) - (h_t+3h_y)/2\cdot(x^2-2x+1) - h_t$
		$\overline{l_0} = (h_t^2+3h_y^2)/4\cdot(x+1) - (h_t+3h_y)/2\cdot(x^2-2x+1) - h_t$
		$\overline{l_1} = (h_t^2+3h_y^2)/4\cdot(x^3-x^2+1) - h_t(x^2-2x+1) - (h_t-3h_y)/2$
	$c_{\mathrm{bw},3}$	$l_0 = (h_t^2+3h_y^2)/4\cdot(x^3-x^2+1) + h_t + (h_t+3h_y)/2\cdot(x-1)^2$
		$l_1 = (h_t^2+3h_y^2)/4\cdot(x+1) - (h_t-3h_y)/2\cdot(x^2-2x+2) + h_t$
		$\overline{l_0} = (h_t^2+3h_y^2)/4\cdot(x+1) - (h_t-3h_y)/2\cdot(x^2-2x+2) + h_t$
		$\overline{l_1} = (h_t^2+3h_y^2)/4\cdot(x^3-x^2+1) + (h_t+3h_y)/2\cdot(x^2-2x+1) + h_t$
$\mathbb{G}_2$	$c'_{\mathrm{bw},0}$	$l_0 = (h_t^2+3h_y^2)/4(x+1) + (h_t+3h_y)/2(x^2-2x+2) - h_t$
		$l_1 = (h_t^2+3h_y^2)/4(x^3-x^2+1) - (h_t-3h_y)/2(x^2-2x+1) - h_t$
		$\overline{l_0} = -(h_t^2+3h_y^2)/4(x+1) - (h_t+3h_y)/2(x^2-2x+2) + h_t$
		$\overline{l_1} = (h_t^2+3h_y^2)/4(x^3-x^2-x) - h_t(x^2-2x+1) - (h_t+3h_y)/2$
	$c'_{\mathrm{bw},3}$	$l_0 = -(h_t^2+3h_y^2)/4(x+1) - (h_t-3h_y)/2(x^2-2x+1) - h_t$
		$l_1 = (h_t^2+3h_y^2)/4(x^3-x^2-x) + (h_t+3h_y)/2(x^2-2x+2) - h_t$
		$\overline{l_0} = (h_t^2+3h_y^2)/4(x+1) + (h_t-3h_y)/2(x^2-2x+1) + h_t$
		$\overline{l_1} = (h_t^2+3h_y^2)/4(x^3-x^2+1) + h_t(x^2-2x+1) + (h_t+3h_y)/2$

Pairing computation: Miller Loop. Assuming an endomorphism of eigenvalue $\lambda_{\mathrm{bw},i} = t_{\mathrm{bw},i} - 1$, the formulas are

$$
\begin{aligned}
-x-1+(x^5-x^4+1)(t_{\mathrm{bw},0}-1) &= -3r_{\mathrm{bw}}((x-1)^2(x^2+1)+1) && (4.26)\\
x^5-x^4-x+(x+1)(t_{\mathrm{bw},0}-1) &= -3r_{\mathrm{bw}} && (4.27)\\
x+1+(x^5-x^4-x)(t_{\mathrm{bw},3}-1) &= 3r_{\mathrm{bw}}(x-1)^2(x^2+1) && (4.28)\\
x^5-x^4+1-(x+1)(t_{\mathrm{bw},3}-1) &= -3r_{\mathrm{bw}} && (4.29)
\end{aligned}
$$

and one obtains optimal ate and Tate (a.k.a. twisted ate) pairings from (4.26), (4.29)

$$
\begin{aligned}
m_{\mathrm{Tate}} &= f_{-(u+1)+(u^5-u^4+1)\lambda_{\mathrm{bw},0},P}(Q), & m_{\mathrm{Tate}} &= f_{u^5-u^4+1-(u+1)\lambda_{\mathrm{bw},3},P}(Q),\\
m_{\mathrm{ate}} &= f_{-(u+1)+(u^5-u^4+1)p_{\mathrm{bw},0},Q}(P), & m_{\mathrm{ate}} &= f_{u^5-u^4+1-(u+1)p_{\mathrm{bw},3},Q}(P).
\end{aligned}
$$

Pairing computation: final Exponentiation. Like for BLS12-BW6, the hard part can be expressed in terms of $p_{\mathrm{bw},i}, h_t, h_y$. One obtains two cases. Note that according to Table 4.5, $(h_t^2+3_y^2)/4$ and $(h_t-h_y)/2$ are integers. With the parameters of Table 4.9, the exponent $(p_{\mathrm{bw},i}^2 - p_{\mathrm{bw},i}+1)/r_{\mathrm{bw}}$ multiplied by $3(x+1)$ has coefficients of low degree in x in basis $p_{\mathrm{bw},i}$. The highest power to compute is u^{15} due to $c_{\mathrm{bw},i}$ of degree 10 in u. The two cases have very similar formulas.

$$
\begin{aligned}
&(-x^5+x^4-1+(x+1)p_{\mathrm{bw},0})3(c_{\mathrm{bw},0}+h_t) + 9(x^4+2(-x^3+x^2-x+1)-p_{\mathrm{bw},0}),\\
&(x(x^4-x^3-1)+(x+1)p_{\mathrm{bw},3})3(c_{\mathrm{bw},3}+h_t) + 9(x^4+2(-x^3+x^2-x)+1+p_{\mathrm{bw},3}).
\end{aligned}
$$

Similarly based on the works of Hayashida, Hayasaka and Teruya [HHT20] and Cai, Hu and Zhao [CHZ22], we improve the hard part computation. The exponents of the hard

Table 4.9: Parameters of a BW6 outer curve with a BLS24 inner curve, with $x \equiv 1 \bmod 3$.

r_{bw}	$p_{\mathrm{bls}} = (x-1)^2/3(x^8 - x^4 + 1) + x$ $(x^{10} - 2x^9 + x^8 - x^6 + 2x^5 - x^4 + x^2 + x + 1)/3$	prime
$\overline{\zeta_6}$	$-x^9 + 3x^8 - 4x^7 + 4x^6 - 3x^5 + 2x^3 - 2x^2 + x - 1$	
ζ_6	$x^9 - 3x^8 + 4x^7 - 4x^6 + 3x^5 - 2x^3 + 2x^2 - x + 2$	
$1/\sqrt{-3}$	$(2x^9 - 6x^8 + 8x^7 - 8x^6 + 6x^5 - 4x^3 + 4x^2 - 2x + 3)/3$	
$t_{\mathrm{bw},0}$	$-x^9 + 3x^8 - 4x^7 + 4x^6 - 3x^5 + 2x^3 - 2x^2 + x$	$6 \mid t_{\mathrm{bw},0}$
$t_{\mathrm{bw},3}$	$x^9 - 3x^8 + 4x^7 - 4x^6 + 3x^5 - 2x^3 + 2x^2 - x + 3$	$3 \mid t_{\mathrm{bw},3}, 2 \nmid t_{\mathrm{bw},3}$
$y_{\mathrm{bw},0}$	$(x^9 - 3x^8 + 4x^7 - 4x^6 + 3x^5 - 2x^3 + 2x^2 - x)/3$	
$y_{\mathrm{bw},0}$	$-t_{\mathrm{bw},0}/3$	$2 \mid y_{\mathrm{bw},0}$
$y_{\mathrm{bw},3}$	$(x^9 - 3x^8 + 4x^7 - 4x^6 + 3x^5 - 2x^3 + 2x^2 - x + 3)/3$	
$y_{\mathrm{bw},3}$	$t_{\mathrm{bw},3}/3$	$2 \nmid y_{\mathrm{bw},3}$
$p_{\mathrm{bw},0}$	$((t_{\mathrm{bw},0} + h_t r_{\mathrm{bw}})^2 + 3(y_{\mathrm{bw},0} + h_y r_{\mathrm{bw}})^2)/4$	prime
$p_{\mathrm{bw},3}$	$((t_{\mathrm{bw},3} + h_t r_{\mathrm{bw}})^2 + 3(y_{\mathrm{bw},3} + h_y r_{\mathrm{bw}})^2)/4$	prime
$\Phi_6(t_{\mathrm{bw},i} - 1)$	$(x^8 - 4x^7 + 8x^6 - 12x^5 + 15x^4 - 14x^3 + 10x^2 - 6x + 3) \cdot 3 \cdot r_{\mathrm{bw}}$	
$c_{\mathrm{bw},0}$	$(h_t^2 + 3h_y^2)/4r_{\mathrm{bw}} + (h_t - h_y)/2t_{\mathrm{bw},0} + \Phi_6(t_{\mathrm{bw},0} - 1)/(3r_{\mathrm{bw}}) - h_t$	
$c_{\mathrm{bw},3}$	$(h_t^2 + 3h_y^2)/4r_{\mathrm{bw}} + (h_t + h_y)/2t_{\mathrm{bw},3} + \Phi_6(t_{\mathrm{bw},3} - 1)/(3r_{\mathrm{bw}}) - h_t$	
$c'_{\mathrm{bw},0}$ $(\mathbb{G}_2)$	$c_{\mathrm{bw},0} + (h_t + 3h_y)/2$	
$c'_{\mathrm{bw},3}$ $(\mathbb{G}_2)$	$c_{\mathrm{bw},3} + (h_t - 3h_y)/2$	
Optimal ate Miller loop $f_{a_0,Q}(P) \cdot f^p_{a_1,Q}(P)$, Optimal twisted ate $f_{a_0+a_1\lambda,P}(Q)$		
$\lambda = t_{\mathrm{bw},0} - 1$	$a_0 = -(x+1),\ a_1 = x^5 - x^4 + 1 \quad a'_0 = x^5 - x^4 - x,\ a'_1 = x + 1$	
$\lambda = t_{\mathrm{bw},3} - 1$	$a_0 = x + 1,\ a_1 = x^5 - x^4 - x \quad a'_0 = x^5 - x^4 + 1,\ a'_1 = -(x+1)$	
Final exponentiation		
$e_{\mathrm{bw},0}$	$(x+1)\Phi_6(p_{\mathrm{bw},0})/r_{\mathrm{bw}} =$ $(-x^5 + x^4 - 1 + (x+1)p_{\mathrm{bw},0})(c_{\mathrm{bw},0} + h_t) + 3(x^4 + 2(-x^3 + x^2 - x + 1) - p_{\mathrm{bw},0})$	
$e'_{\mathrm{bw},0}$	$(x^5 - x^4 - x)\Phi_6(p_{\mathrm{bw},0})/r_{\mathrm{bw}} =$ $((x+1) + (x^5 - x^4 - x)p_{\mathrm{bw},0})(c_{\mathrm{bw},0} + h_t) - 3(1 + (x^4 - 2x^3 + 2x^2 - 2x + 1)p_{\mathrm{bw},0})$	
$e_{\mathrm{bw},3}$	$(x+1)\Phi_6(p_{\mathrm{bw},3})/r_{\mathrm{bw}} =$ $(x(x^4 - x^3 - 1) + (x+1)p_{\mathrm{bw},3})(c_{\mathrm{bw},3} + h_t) + 3(x^4 + 2(-x^3 + x^2 - x) + 1 + p_{\mathrm{bw},3})$	
$e'_{\mathrm{bw},3}$	$(x^5 - x^4 + 1)\Phi_6(p_{\mathrm{bw},3})/r_{\mathrm{bw}} =$ $((-x-1) + (x^5 - x^4 + 1)p_{\mathrm{bw},3})(c_{\mathrm{bw},3} + h_t) - 3(1 - (x^4 - 2x^3 + 2x^2 - 2x + 2)p_{\mathrm{bw},3})$	

part of the final exponentiation for BW6-BLS24 curves are the following.

$$
\begin{aligned}
c_0 &= (x-1)^2(x^2+1)\\
(x^5-x^4-x)\frac{\Phi_6(p_0)}{r} &= \Big(\overbrace{(x+1)\underbrace{(1+c_0p)}_{a_0}-p}^{b_0}\Big)\Big(\tfrac{h_t^2+3h_y^2}{4}r+3\big(\tfrac{h_t-h_y}{2}\tfrac{t_0}{3}+\tfrac{d-1}{3}\big)+1\Big)-3\underbrace{(1+c_0p)}_{a_0}\\
c_0' &= (x-1)^2(x^2+1)+1 = c_0+1\\
(x+1)\frac{\Phi_6(p_0)}{r} &= \Big(\overbrace{(x+1)\underbrace{(p-c_0')}_{a_0'}+1}^{b_0'}\Big)\Big(\tfrac{h_t^2+3h_y^2}{4}r+3\big(\tfrac{h_t-h_y}{2}\tfrac{t_0}{3}+\tfrac{d-1}{3}\big)+1\Big)-3\underbrace{(p-c_0')}_{a_0}
\end{aligned}
$$

$$
\begin{aligned}
c_3 &= (x-1)^2(x^2+1)\\
(x+1)\frac{\Phi_6(p_3)}{r} &= \Big(\overbrace{(x+1)\underbrace{(c_3+p)}_{a_3}-1}^{b_3}\Big)\Big(\tfrac{h_t^2+3h_y^2}{4}r+3\big(\tfrac{h_t+h_y}{2}\tfrac{t_3}{3}+\tfrac{d-1}{3}\big)+1\Big)+3\underbrace{(c_3+p)}_{a_3}\\
c_3' &= (x-1)^2(x^2+1)+1 = c_3+1\\
(x^5-x^4+1)\frac{\Phi_6(p_3)}{r} &= \Big(\overbrace{(x+1)\underbrace{(c_3'p-1)}_{a_3'}-p}^{b_3'}\Big)\Big(\tfrac{h_t^2+3h_y^2}{4}r+3\big(\tfrac{h_t+h_y}{2}\tfrac{t_3}{3}+\tfrac{d-1}{3}\big)+1\Big)+3\underbrace{(c_3'p-1)}_{a_3'}
\end{aligned}
$$

The parameters are given in Tab. 4.9, and we set $d = \Phi_6(t_i-1)/(3r) = x^8-4x^7+8x^6-12x^5+15x^4-14x^3+10x^2-6x+3$. Part of the exponent is

$$
\begin{aligned}
&\frac{h_t^2+3h_y^2}{4}r+\frac{h_t+h_y}{2}t_3+d\\
&\frac{h_t^2+3h_y^2}{4}r+\frac{h_t-h_y}{2}t_0+d
\end{aligned}
$$

We compute the exponents r, $t_0/3$, $t_3/3$ and $(d-1)/3$ as follows and obtain Alg. 4.5 and Alg. 4.6.

$$
\begin{array}{lcll}
a &=& (x-1)/3 & \\
b &=& a(x-1)(x^2+1) & \\
c &=& b((x-1)^2(x^2+1)+1) &= (d-1)/3\\
 & & & \\
f_0 &=& -(x+1)c+b-a &= t_0/3\\
g &=& -(x+1)(f_0+b)+a+1 &= r
\end{array}
\qquad
\begin{array}{lcll}
a &=& (x-1)/3 & \\
b &=& a(x-1)(x^2+1) & \\
c &=& b((x-1)^2(x^2+1)+1) &= (d-1)/3\\
e_3 &=& (x+1)c-b+a & \\
f_3 &=& e_3+1 &= t_3/3\\
g &=& (x+1)(e_3-b)+a+1 &= r
\end{array}
$$

Algorithm 4.5: Hard part of final exp., BLS24-BW6, t_0
Input: x, m, $h_1 = (h_t - h_y)/2$, $h_2 = (h_t^2 + 3h_y^2)/4$
Output: $m^{(x^5-x^4-x)\Phi_6(p_0)/r}$

$m_p \leftarrow m^p$
$A \leftarrow (m_p^{u-1})^{u-1}$
$A \leftarrow A^{u^2+1}$ ▷or $(A^u)^u \cdot A$
$A \leftarrow m \cdot A$
$B \leftarrow A^{u+1} \cdot \overline{m_p}$ ▷m^{b_0}
$A \leftarrow \overline{A^2 \cdot A}$ ▷m^{-3a_0}
$C \leftarrow B^{(u-1)/3}$
$D \leftarrow C^{u-1}$
$D \leftarrow D^{u^2+1}$ ▷or $(D^u)^u \cdot D$
$E \leftarrow (D^{u-1})^{u-1}$
$E \leftarrow E^{u^2+1}$ ▷or $(E^u)^u \cdot E$
$E \leftarrow E \cdot D$ ▷$B^{(d-1)/3}$
$F \leftarrow \overline{E^{u+1} \cdot C} \cdot D$ ▷$B^{t_0/3}$
$G \leftarrow \overline{F \cdot D}$
$H \leftarrow G^{u+1} \cdot C \cdot B$ ▷B^r
$I \leftarrow F^{d_1} \cdot E$
$I \leftarrow I^2 \cdot I \cdot B \cdot H^{d_2}$
return $A \cdot I$

Algorithm 4.6: Hard part of final exp., BLS24-BW6, t_3
Input: x, m, $h_1 = (h_t + h_y)/2$, $h_2 = (h_t^2 - 3h_y^2)/4$
Output: $m^{(x+1)\Phi_6(p_3)/r}$

$A \leftarrow (m^{u-1})^{u-1}$
$A \leftarrow A^{u^2+1}$ ▷or $(A^u)^u \cdot A$
$A \leftarrow A \cdot m^p$
$B \leftarrow A^{u+1} \cdot \overline{m}$ ▷m^{b_3}
$A \leftarrow A^2 \cdot A$ ▷m^{3a_3}
$C \leftarrow B^{(u-1)/3}$
$D \leftarrow C^{u-1}$
$D \leftarrow D^{u^2+1}$ ▷or $(D^u)^u \cdot D$
$E \leftarrow (D^{u-1})^{u-1}$
$E \leftarrow E^{u^2+1}$ ▷or $(E^u)^u \cdot E$
$E \leftarrow E \cdot D$ ▷$B^{(d-1)/3}$
$D \leftarrow \overline{D}$
$F \leftarrow E^{u+1} \cdot D \cdot C$
$G \leftarrow F \cdot B$ ▷$B^{t_3/3}$
$H \leftarrow (F \cdot D)^{u+1} \cdot C \cdot B$ ▷B^r
$I \leftarrow G^{d_1} \cdot E$
$I \leftarrow I^2 \cdot I \cdot B \cdot H^{d_2}$
return $A \cdot I$

BW6 with BN

For completeness, we generalize the 2-chain framework to include BN curves as inner curves. This rediscovers the BN/BW6 2-chain from the Geppetto paper. Table 4.10 gives the parameters of a BW6 outer curve with a BN inner curve for any integer x.

Pairing computation: final Exponentiation. A lattice reduction (with Magma on polynomials) gives the formulas in Tab. 4.10, with the precomputation $d = \Phi_6(t_i - 1)/(3r) = 3x(x+1)+1$. We highlight with an underbrace the similar parts that can be shared. The sequential steps are $t_0 = -3x(2d+1)$ then $r = 2(-xt_0 + d) - 1$ in Alg. 4.7,

Table 4.10: Parameters of a BW6 outer curve with a BN inner curve, any integer x.

r_{bw}	$p_{\mathrm{bn}} = 36x^4 + 36x^3 + 24x^2 + 6x + 1$	prime
$\overline{\zeta_6}$	$-18x^3 - 18x^2 - 9x - 1$	
ζ_6	$18x^3 + 18x^2 + 9x + 2$	
$1/\sqrt{-3}$	$12x^3 + 12x^2 + 6x + 1$	
$t_{\mathrm{bw},0}$	$-18x^3 - 18x^2 - 9x$	$9 \mid t_{\mathrm{bw},0}$
$t_{\mathrm{bw},3}$	$18x^3 + 18x^2 + 9x + 3$	$3 \mid t_{\mathrm{bw},3}$
$y_{\mathrm{bw},0}$	$-t_{\mathrm{bw},0}/3 = 6x^3 + 6x^2 + 3x$	$3 \mid y_{\mathrm{bw},0}$
$y_{\mathrm{bw},3}$	$t_{\mathrm{bw},3}/3 = 6x^3 + 6x^2 + 3x + 1$	
$p_{\mathrm{bw},0}$	$((t_{\mathrm{bw},0} + h_t r_{\mathrm{bw}})^2 + 3(y_{\mathrm{bw},0} + h_y r_{\mathrm{bw}})^2)/4$	prime
$p_{\mathrm{bw},3}$	$((t_{\mathrm{bw},3} + h_t r_{\mathrm{bw}})^2 + 3(y_{\mathrm{bw},3} + h_y r_{\mathrm{bw}})^2)/4$	prime
$\Phi_6(t_{\mathrm{bw},i} - 1)$	$(9x^2 + 9x + 3) \cdot r_{\mathrm{bw}}$	
$c_{\mathrm{bw},0}$	$(h_t^2 + 3h_y^2)/4r_{\mathrm{bw}} + (h_t - h_y)/2t_{\mathrm{bw},0} + \Phi_6(t_{\mathrm{bw},0} - 1)/(3r_{\mathrm{bw}}) - h_t$	
$c_{\mathrm{bw},3}$	$(h_t^2 + 3h_y^2)/4r_{\mathrm{bw}} + (h_t + h_y)/2t_{\mathrm{bw},3} + \Phi_6(t_{\mathrm{bw},3} - 1)/(3r_{\mathrm{bw}}) - h_t$	
$c'_{\mathrm{bw},0}$ $(\mathbb{G}_2)$	$c_{\mathrm{bw},0} + (h_t + 3h_y)/2$	
$c'_{\mathrm{bw},3}$ $(\mathbb{G}_2)$	$c_{\mathrm{bw},3} + (h_t - 3h_y)/2$	
Optimal ate Miller loop $f_{a_0,Q}(P) \cdot f^p_{a_1,Q}(P)$, Optimal twisted ate $f_{a_0+a_1\lambda,P}(Q)$		
$\lambda = t_{\mathrm{bw},0} - 1$	$a_0 = 2x,\ a_1 = 6x^2 + 2x + 1 \quad a'_0 = 6x^2 + 4x + 1,\ a'_1 = -2x$	
$\lambda = t_{\mathrm{bw},3} - 1$	$a_0 = 6x^2 + 2x + 1,\ a_1 = 2x \quad a'_0 = -2x,\ a'_1 = 6x^2 + 4x + 1$	
Final exponentiation		
$e_{\mathrm{bw},0}$	$2x\Phi_6(p_{\mathrm{bw},0})/r_{\mathrm{bw}} =$ $(6x^2 + 2x + 1 + 2xp_{\mathrm{bw},0})(c_{\mathrm{bw},0} + h_t) - (3x + 1 + p_{\mathrm{bw},0})$	
$e'_{\mathrm{bw},0}$	$(6x^2 + 4x + 1)\Phi_6(p_{\mathrm{bw},0})/r_{\mathrm{bw}} =$ $(-2x + (6x^2 + 4x + 1)p_{\mathrm{bw},0})(c_{\mathrm{bw},0} + h_t) + 1 - (3x + 2)p_{\mathrm{bw},0}$	
$e_{\mathrm{bw},3}$	$2x\Phi_6(p_{\mathrm{bw},3})/r_{\mathrm{bw}} =$ $(-(6x^2 + 4x + 1) + 2xp_{\mathrm{bw},3})(c_{\mathrm{bw},3} + h_t) - (3x + 2) + p_{\mathrm{bw},3}$	
$e'_{\mathrm{bw},3}$	$(6x^2 + 2x + 1)\Phi_6(p_{\mathrm{bw},3})/r_{\mathrm{bw}} =$ $(2x + (6x^2 + 2x + 1)p_{\mathrm{bw},3})(c_{\mathrm{bw},3} + h_t) + 1 + (3x + 1)p_{\mathrm{bw},3}$	

and $t_3 = 3(x(2d+1)+1)$, then $r = 2x(t_3+3x)+1$ in Alg. 4.8.

$$(2x)\frac{\Phi_6(p_0)}{r} = \overbrace{\big(1+2x\underbrace{(3x+1+p_0)}_{a_0}\big)}^{b_0}\Big(\tfrac{h_t^2+3h_y^2}{4}r + \tfrac{h_t-h_y}{2}t_0 + d\Big) - \underbrace{(3x+1+p_0)}_{a_0}$$

$$(6x^2+4x+1)\frac{\Phi_6(p_0)}{r} = \overbrace{\big(p_0-2x\underbrace{(1-(3x+2)p_0)}_{a'_0}\big)}^{b'_0}\Big(\tfrac{h_t^2+3h_y^2}{4}r + \tfrac{h_t-h_y}{2}t_0 + d\Big) + \underbrace{1-(3x+2)p_0}_{a'_0}$$

$$(2x)\frac{\Phi_6(p_3)}{r} = \overbrace{\big(2x\underbrace{(p_3-(3x+2))}_{a_3}-1\big)}^{b_3}\Big(\tfrac{h_t^2+3h_y^2}{4}r + \tfrac{h_t+h_y}{2}t_3 + d\Big) + \underbrace{p_3-(3x+2)}_{a_3}$$

$$(6x^2+2x+1)\frac{\Phi_6(p_3)}{r} = \overbrace{\big(2x\underbrace{(1+(3x+1)p_3)}_{a'_3}+p_3\big)}^{b'_3}\Big(\tfrac{h_t^2+3h_y^2}{4}r + \tfrac{h_t+h_y}{2}t_3 + d\Big) + \underbrace{1+(3x+1)p_3}_{a'_3}$$

Algorithm 4.7: Hard part of final exp., BN-BW6, $t_{\mathrm{bn},0}$

Input: x, m, $h_1 = (h_t - h_y)/2$, $h_2 = (h_t^2 + 3h_y^2)/4$

Output: $m^{(2x)\Phi_6(p_0)/r}$

1 $A \leftarrow m^x$
2 $B \leftarrow A^2 \cdot A \cdot m \cdot m^p$ $\triangleright m^{a_0}$
3 $A \leftarrow \overline{B}$
4 $B \leftarrow (B^2)^x \cdot m$ $\triangleright m^{b_0}$
5 $C \leftarrow (B^x \cdot B)^x$
6 $C \leftarrow C^2 \cdot C \cdot B$ $\triangleright B^d$
7 $D \leftarrow (C^2 \cdot B)^x$
8 $D \leftarrow D^2 \cdot D$ $\triangleright B^{-t_0}$
9 $F \leftarrow (D^x \cdot C)^2 \cdot \overline{F}$ $\triangleright B^r$
10 $F \leftarrow F^{h_2} \cdot \overline{D}^{h_1} \cdot C$
11 **return** $F \cdot A$

Algorithm 4.8: Hard part of final exp., BN-BW6, $t_{\mathrm{bn},3}$

Input: x, m, $h_1 = (h_t + h_y)/2$, $h_2 = (h_t^2 + 3h_y^2)/4$

Output: $m^{(6x^2+2x+1)\Phi_6(p_3)/r}$

1 $B \leftarrow m^p$
2 $A \leftarrow B^x$
3 $A \leftarrow (A^2 \cdot A \cdot B) \cdot m$ $\triangleright m^{a'_3}$
4 $B \leftarrow (A^2)^x \cdot B$ $\triangleright m^{b'_3}$
5 $C \leftarrow B^x$
6 $D \leftarrow C^2 \cdot C$
7 $C \leftarrow D^x \cdot D \cdot B$ $\triangleright B^d$
8 $E \leftarrow (C^2 \cdot B)^x \cdot B$
9 $E \leftarrow E^2 \cdot E$ $\triangleright B^{t_3}$
10 $F \leftarrow ((E \cdot D)^x)^2 \cdot B$ $\triangleright B^r$
11 $F \leftarrow F^{h_2} \cdot E^{h_1} \cdot C$
12 **return** $F \cdot A$

Two-chains with inner BLS and outer Cocks-Pinch

Section 21 showed that a Brezing–Weng outer curve of embedding degree $k = 6$ is optimal with a BLS-12 curve whose prime-order subgroup is about 256 bits long. However BW6 is no longer optimal with BLS24 over a prime field of about 320 bits: we measure the security in the finite field $\mathbb{F}_{p^6}$ whose p is roughly 640 bits long to be about 124 bits in Section 4.5.2. To increase the security in the finite field $\mathbb{F}_{p^k}$, we can increase the size of the prime p thanks to the choice of lifting co-factors h_t, h_y, and obtain a p of 672 bits, or we can increase the embedding degree k, but then the BW construction is no longer available: we move to the Cocks-Pinch construction. To allow twist optimisation, we focus on $k = 8$ with $D = 1$ (quartic twist) and $k = 12$ with $D = 3$ (sextic twist). Our Cocks-Pinch curves are similar to the curves of Guillevic, Masson and Thomé [GMT20] (see also [Mas20, Chapter 5]). The lifting cofactor idea appeared before in Fotiadis and Konstantinou paper [FK19].

With the Cocks-Pinch construction of embedding degree not 6, the optimal ate pairing like for BW6 curves is no longer available because the eigenvalue of the Frobenius endomorphism π_p on a CP curve $E(\mathbb{F}_{p^k})$ does not have a simple polynomial form modulo the subgroup order $r_\times = p_{\text{bls}}$. In other words, there is no k-th root of unity modulo $p_{\text{bls}}(x)$ (as polynomials). However, π_p has an eigenvalue (as a scalar integer) modulo $r_\times(u) \in \mathbb{Z}$, and one can use a lattice basis reduction algorithm (like LLL) to obtain a decomposition with short scalars a_i, of size roughly $r_\times^{1/4}$: $a_0 + a_1 p_\times + a_2 p_\times^2 + a_3 p_\times^3 = 0 \bmod r_\times$. This 4-fold holds for CP8 and CP12 curves as $\varphi(8) = \varphi(12) = 4$. The optimal ate Miller loop would be

$$f_{a_0,Q}(P) f_{a_1,Q}^{p}(P) f_{a_2,Q}^{p^2}(P) f_{a_3,Q}^{p^3}(P) \ell_{a_0 Q, a_1 \pi_p(Q)}(P) \ell_{a_2 \pi_{p^2}(Q), a_3 \pi_{p^3}(Q)} \ .$$

But the scalars a_i are not sparse and none of them is trivial, contrary to [Ver10]. Instead, we generalize our Alg. 4.2 and obtain Alg. 4.10. Algorithm 4.9 precomputes the data and Alg. 4.10 computes the pairing, with the formulas (4.11)–(4.14) adapted to the ate pairing with swapped P and Q and $\lambda = p$, and with $C_i = \sum_i c_i p^i$,

$$f_{2C_i,Q}(P) = f_{C_i,Q}^2(P) \ell_{[C_i]Q,[C_i]Q}(P) \tag{4.30}$$

$$\begin{aligned} &f_{C_i+p^j+p^l+p^m,Q}(P) = f_{C_i,Q}(P) f_{p^j+p^l+p^m,Q}(P) \ell_{[C_i]Q,[p^j+p^l+p^m]Q}(P) \\ &= f_{C_i,Q}(P) \ell_{[C_i]Q,[p^j+p^l+p^m]Q}(P) \ell_{[p^j+p^l]Q,[p^m]Q}(P) \ell_{[p^j]Q,[p^l]Q}(P) \end{aligned} \tag{4.31}$$

$$\begin{aligned} &f_{C_i+1+p+p^2+p^3,Q}(P) = f_{C_i,Q}(P) f_{1+p+p^2+p^3,Q}(P) \ell_{[C_i]Q,[1+p+p^2+p^3]Q}(P) \\ &= f_{C_i,Q}(P) \ell_{[C_i]Q,[1+p+p^2+p^3]Q}(P) \\ &\cdot \ell_{[1+p]Q,[p^2+p^3]Q}(P) \ell_{Q,[p]Q}(P) \ell_{[p^2]Q,[p^3]Q}(P) \end{aligned} \tag{4.32}$$

The $f_{p^j,Q}(P)$ terms are removed [HSV06]. The points $[p^j]Q$, $[p^j+p^l]Q$, $[p^j+p^l+p^m]Q$, $[1+p+p^2+p^3]Q$, lines $\ell_{[p^m]Q,[p^n]Q}(P)$, $\ell_{[p^j+p^l]Q,[p^m]Q}(P)$, $\ell_{[1+p]Q,[p^2+p^3]Q}(P)$, and their products, are precomputed.

On CP8 curves, $\mathbb{G}_1$ has an endomorphism $\phi\colon (x,y) \mapsto (-x, \sqrt{-1}y)$ of eigenvalue $\lambda \equiv p^2 \bmod r$, $\lambda^2 \equiv -1 \bmod r$. On CP12 curves, $\mathbb{G}_1$ has the same endomorphism as BW6 curves, of eigenvalue $\lambda \equiv p^2 \bmod r$. The twisted ate pairing on our CP curves has Miller loop $f_{\lambda,P}(Q) = f_{p^2,P}(Q)$, and we derive our optimal Tate pairing like for BW6 curves, with short scalars $a_0 + a_1\lambda \equiv 0 \bmod r$.

Comparison of BW6, CP8 and CP12 outer curve performances

We reproduce the field arithmetic estimates from [GMT20, EHG20] in Table 4.11 and the pairing cost estimates in Table 4.12. Parameters of CP8 and CP12 are in Table 4.13, BW6 are in Table 4.17. We justify our choice of seeds and curve parameters in Section 4.5. Ate and Tate pairing estimates of our BW6 and CP curves are in Table 4.14. We have a speed-up of the optimal ate pairing on BW6 curves compared to [EHG20] with the formula (4.33) with $v = u^2 - 2u + 1$ for BLS12-BW6 and $v = u^4 - 2(u^3 - u^2 + u) + 1$ for BLS24-BW6 because the 2-NAF Hamming weight of the scalar v is lower.

$$f_{u+1} = f_{u+1,Q}(P), \quad m_{\text{opt. ate}} = (f_{u+1})_{v,[u+1]Q}^{p}(P) \ell_{[(u+1)v]Q,-Q}^{p}(P) \ . \tag{4.33}$$

BW6 curves as outer curves of BLS24 have a pairing faster than CP8 and CP12 curves: *a larger characteristic gives better performances than a larger embedding degree.* Assuming a ratio $\mathbf{m}_{704}/\mathbf{m}_{640} = 1.25$, an ate Miller loop on CP8-632 is 25% slower compared to BW6-672, but the final exponentiation is 15% faster. A full pairing on CP8 is about 7% slower, and 59% slower on CP12. BLS24-BW6 has a faster pairing than BLS12-BW6, but the 2-adicity of BLS24 curves is much smaller.

Algorithm 4.9: Precomputations of sums of points and lines
Input: $P \in E(\mathbb{F}_p)[r]$, $Q_0, Q_1, Q_2, Q_3 \in E'(\mathbb{F}_{p^{k/d}})[r]$
Output: array T of length 15, of precomputed points and lines

$T \leftarrow$ array of length 15
for $i = 0$ *to* 3**:**
    $T[2^i - 1][0] \leftarrow Q_i$; $T[2^i - 1][1] \leftarrow 1$
for $0 \leq m < n \leq 3$**:**
    $i \leftarrow 2^m + 2^n$
    $T[i-1][0] \leftarrow T[2^m - 1] + T[2^n - 1]$
    $T[i-1][1] \leftarrow \ell_{Q_m, Q_n}(P)$
for $0 \leq m < n < s \leq 3$**:**
    $i \leftarrow 2^m + 2^n + 2^s$
    $T[i-1][0] \leftarrow T[2^m + 2^n - 1][0] + T[2^s - 1][0]$
    $T[i-1][1] \leftarrow T[2^m + 2^n - 1][1] \cdot \ell_{Q_m + Q_n, Q_s}(P)$
$T[15-1][0] \leftarrow T[7-1][0] + T[8-1]$
$T[15-1][1] \leftarrow T[7-1][1] \cdot \ell_{Q_0+Q_1+Q_2, Q_3}(P)$
return T

Algorithm 4.10: Miller loop for optimal ate pairing, Cocks-Pinch
Input: $P \in \mathbb{G}_1 = E(\mathbb{F}_p)[r]$, $Q \in \mathbb{G}_2 = \ker(\pi_p - [p]) \cap E(\mathbb{F}_{p^k})[r]$, scalars a_0, a_1, a_2, a_3 such that $a_0 + a_1 p + a_2 p^2 + a_3 p^3 = 0 \bmod r$
Output: $f_{a_0 + a_1 p + a_2 p^2 + a_3 p^3, Q}(P)$

$Q_0 \leftarrow Q$; $Q_1 \leftarrow \pi_p(Q)$; $Q_2 \leftarrow \pi_{p^2}(Q)$; $Q_3 \leftarrow \pi_{p^3}(Q)$
for $i = 0$ *to* 3**:**
    if $a_i < 0$**:** $a_i \leftarrow -a_i$; $Q_i \leftarrow -Q_i$
$T \leftarrow$ precomputations(Q_0, Q_1, Q_2, Q_3)
$l_i \leftarrow bits(a_i)$ for $0 \leq i \leq 3$
$i \leftarrow \max_{0 \leq j \leq 3} (\operatorname{len} l_j)$
$j \leftarrow l_{0,i} + 2l_{1,i} + 4l_{2,i} + 8l_{3,i}$
$f \leftarrow T[j-1][1]$
$S \leftarrow T[j-1][0]$
for $i = i - 1$ *downto* 0**:**
    $f \leftarrow f^2$
    $\ell_t \leftarrow \ell_{S,S}(P)$; $S \leftarrow [2]S$
    $j \leftarrow l_{0,i} + 2l_{1,i} + 4l_{2,i} + 8l_{3,i}$
    if $j > 0$**:**
        $Q_j \leftarrow T[j-1][0]$; $\ell \leftarrow \ell_{S,Q_j}(P)$; $S \leftarrow S + Q_j$
        $f \leftarrow f \cdot (\ell_t \cdot \ell)$
        if $T[j-1][1] \neq 1$**:** $f \leftarrow f \cdot T[j-1][1]$
    else: $f \leftarrow f \cdot \ell_t$
return f

Table 4.11: Cost from [GMT20, Tab. 6] of $\mathbf{m}_k$, $\mathbf{s}_k$ and $\mathbf{i}_k$ for field extensions $\mathbb{F}_{p^k}$. Inversions in $\mathbb{F}_{p^{ik}}$ come from $\mathbf{i}_{2k} = 2\mathbf{m}_k + 2\mathbf{s}_k + \mathbf{i}_k$ and $\mathbf{i}_{3k} = 9\mathbf{m}_k + 3\mathbf{s}_k + \mathbf{i}_k$. $\mathbb{F}_{p^{12}}$, resp. $\mathbb{F}_{p^{24}}$ always have a first quadratic, resp. quartic extension, $\mathbf{i}_{24} = 2\mathbf{m}_{12} + 2\mathbf{s}_{12} + \mathbf{i}_{12} = 293\mathbf{m} + \mathbf{i}$ with $\mathbf{i}_{12} = 9\mathbf{m}_4 + 3\mathbf{s}_4 + \mathbf{i}_4$, and for $\mathbb{F}_{p^{12}}$, $\mathbf{i}_{12} = 2\mathbf{m}_6 + 2\mathbf{s}_6 + \mathbf{i}_6 = 97\mathbf{m} + \mathbf{i}$ with $\mathbf{i}_6 = 9\mathbf{m}_2 + 3\mathbf{s}_2 + \mathbf{i}_2$.

k	1	2	3	4	6	8	12	24
$\mathbf{m}_k$	$\mathbf{m}$	$3\mathbf{m}$	$6\mathbf{m}$	$9\mathbf{m}$	$18\mathbf{m}$	$27\mathbf{m}$	$54\mathbf{m}$	$162\mathbf{m}$
$\mathbf{s}_k$	$\mathbf{m}$	$2\mathbf{m}$	$5\mathbf{m}$	$6\mathbf{m}$	$12\mathbf{m}$	$18\mathbf{m}$	$36\mathbf{m}$	$108\mathbf{m}$
$\mathbf{f}_k$	0	0	$2\mathbf{m}$	$2\mathbf{m}$	$4\mathbf{m}$	$6\mathbf{m}$	$10\mathbf{m}$	$22\mathbf{m}$
$\mathbf{s}_k^{\text{cyclo}}$	–	$2\mathbf{s}$	–	$4\mathbf{m}$	$6\mathbf{m}$	$12\mathbf{m}$	$18\mathbf{m}$	$54\mathbf{m}$
$\mathbf{i}_k - \mathbf{i}_1$	0	$2\mathbf{m} + 2\mathbf{s}$	$9\mathbf{m} + 3\mathbf{s}$	$14\mathbf{m}$	$34\mathbf{m}$	$44\mathbf{m}$	$97\mathbf{m}$	$293\mathbf{m}$
$\mathbf{i}_k$, with $\mathbf{i}_1 = 25\mathbf{m}$	$25\mathbf{m}$	$29\mathbf{m}$	$37\mathbf{m}$	$39\mathbf{m}$	$59\mathbf{m}$	$69\mathbf{m}$	$119\mathbf{m}$	$318\mathbf{m}$

Table 4.12: Miller loop cost in non-affine, Weierstrass model [CLN10, AKL+11]. For $6 \mid k$, two sparse-dense multiplications cost $26\mathbf{m}_{k/6}$ whereas one sparse-sparse and one multiplication cost $6\mathbf{m}_{k/6}+\mathbf{m}_k = 24\mathbf{m}_{k/6}$. For $4 \mid k$, this is $16\mathbf{m}_{k/4}$ compared to $6\mathbf{m}_{k/4}+\mathbf{m}_k = 15\mathbf{m}_{k/4}$.

k	D	curve	DoubleLine and AddLine	ref	SparseM and SparseSparseM
$6 \mid k$	-3	$Y^2 = X^3 + b$ sextic twist	$3\mathbf{m}_{k/6} + 6\mathbf{s}_{k/6} + (k/3)\mathbf{m}$ $11\mathbf{m}_{k/6} + 2\mathbf{s}_{k/6} + (k/3)\mathbf{m}$	[AKL+11, §4]	$13\mathbf{m}_{k/6}$ $6\mathbf{m}_{k/6}$
$4 \mid k$	-1	$Y^2 = X^3 + ax$ quartic twist	$2\mathbf{m}_{k/4} + 8\mathbf{s}_{k/4} + (k/2)\mathbf{m}$ $9\mathbf{m}_{k/4} + 5\mathbf{s}_{k/4} + (k/2)\mathbf{m}$	[CLN10, §4]	$8\mathbf{m}_{k/4}$ $6\mathbf{m}_{k/4}$

Table 4.13: CP8 and CP12 outer curve parameters on top of BLS24-315

outer curve	u	(h_t, h_y)	$(t-1)^2 + 1 \bmod r, u$	equation	$\mathbb{F}_{p^k}$ (bits)	est. DL in $\mathbb{F}_{p^k}$
BLS24-315-CP8-632	`-0xbfcfffff`	(6,2)	–	$y^2 = x^3 - x$	5056	140
BLS24-315-CP12-630	`-0xbfcfffff`	(1,2)	0	$y^2 = x^3 - 1$	7560	166

Table 4.14: Pairing cost estimates on BLS12-BW6, BLS24-BW6, BLS24-CP8, BLS24-CP12 curves. BLS12-BW6 curves use Eq. (4.21) with [EHG20, Alg. 5], and $v = u^2 - 2u + 1$. BLS24-BW6 curves use Eq (4.27), (4.28) with $v = u^4 - 2(u^3 - u^2 + u) + 1$.

	BLS12-377-BW6-761	BLS12-379-BW6-764
ate $f_{u+1,Q}(f_u)^p_{u^2-u-1,[u]Q}$	$7863\mathbf{m}_{768}$	$7653\mathbf{m}_{768}$
ate $f_{u+1,Q}(f_{u+1})^p_{v,[u+1]Q}\ell^p_{(u+1)vQ,-Q}$	$7555\mathbf{m}_{768}$	$7389\mathbf{m}_{768}$
Tate $f_{u+1+(u^3-u^2-u)\lambda,P}$ Alg. 4.2	$7729\mathbf{m}_{768}$	$7540\mathbf{m}_{768}$
Final exp. [EHG20, § 3.3, Tab. 7]	$5081\mathbf{m}_{768}$	–
Final exp. Eq. (4.25)	$5195\mathbf{m}_{768}$	$5033\mathbf{m}_{768}$
	BLS24-315-BW6-633	BLS24-315-BW6-672
ate $f_{u+1,Q}(f_{u+1})^p_{v,[u+1]Q}\ell^p_{(u+1)vQ,-Q}$	$7285\mathbf{m}_{640}$	$7285\mathbf{m}_{704}$
Tate $f_{u+1+(u^5-u^4-u)\lambda,P}$ Alg. 4.2	$6813\mathbf{m}_{640}$	$6813\mathbf{m}_{704}$
Final exp.	$5027\mathbf{m}_{640}$	$5501\mathbf{m}_{704}$
	BLS24-315-CP8-632	BLS24-315-CP12-630
ate $f_{a_0+a_1p+a_2p^2+a_3p^3,Q}$ Alg. 4.10	$10679\mathbf{m}_{640}$	$13805\mathbf{m}_{640}$
Tate $f_{a_0+a_1\lambda,P}$ Alg. 4.2	$12489\mathbf{m}_{640}$	$15780\mathbf{m}_{640}$
Final exp.	$5835\mathbf{m}_{640}$	$10312\mathbf{m}_{640}$

4.5 Implementation and benchmarking

In previous sections, we presented families of SNARK-friendly 2-chains that are suitable for Groth16 and KZG-based universal SNARKs. These families are composed of BLS12 and BLS24 inner curves and BW6, CP8 and CP12 outer curves. We demonstrated that the pair family BLS12/BW6 is suitable for recursive Groth16 applications and meets the best security/performance trade-off. Similarly, we showed that BLS24/BW6 is suitable for KZG-based universal SNARKs. We also investigated the family pairs BLS24/CP8 and BLS24/CP12 as more conservative choices and showed that CP8-632 is competitive with BLS24/BW6-672. BW6-633, CP8 and CP12 are defined over a base field of roughly the same bit length, and all have a GLV endomorphism, hence performances on $\mathbb{G}_1$ are expected to be the same. On $\mathbb{G}_2$, BW6 are always faster because they are defined over the same base field as $\mathbb{G}_1$, contrary to CP curves. For the pairing computation, as discussed in Section 19, CP8 and CP12 are slower than both choices of BW6. Additionally, multi-pairings (as used in SNARKs) scale better on BW6 curves (Alg. 4.2) compared to CP8 and CP12. Therefore, we have chosen to focus our benchmarks on BLS12/BW6 and BLS24/BW6 families of curves.

In this section, we first present an open-sourced SageMath library to derive these curves and test our generic formulas. Then, based on additional practical criteria, we recommend a short list of SNARK-friendly 2-chains. Finally, we implement this short-list in the open-sourced `gnark` ecosystem [BPH+22a]. We benchmark the relevant curve operations in $\mathbb{G}_1$ and $\mathbb{G}_2$, and the pairings, and compare efficiency of all choices in practical Groth16 and PLONK settings, which is a popular KZG-based universal SNARK. Both schemes are implemented in `gnark` and maintained by ConsenSys.

4.5.1 SageMath library: derive the curves

At `https://gitlab.inria.fr/zk-curves/snark-2-chains`, we present SageMath scripts to derive all the SNARK-friendly 2-chain families and verify the formulae presented in sections 4.4.1 and 4.4.2, and the pairing cost estimates of Table 4.14.

4.5.2 Our short-list of curves

For all curves, in addition to SNARK-friendliness and security level λ, we shall consider the following properties:

- A seed u with low Hamming weight $\mathrm{HW}(u)$, allowing fast Miller loops in pairings.
- Isogenies of low degree d from a curve with j-invariant different from 0 and 1728, allowing use of the "Simplified Shallue-van de Woestijne-Ulas (SSWU)" method for hashing to the curve [WB19].
- Small integer α relatively prime to $r-1$, allowing the use of x^α as an S-box in the algebraic SNARK-hashes (e.g. Poseidon [GKR+21]).
- Small non-residues in $\mathbb{F}_p$, for an efficient tower arithmetic.
- "Spare" bits in $\mathbb{F}_p$, for carries, infinity point or compressed point flag.
- "Spare" bits in $\mathbb{F}_r$, for optimizing the MSM algorithm (cf. Alg. 5.2 in Chapter 5).

Table 4.15: Seeds of SNARK-friendly inner BLS12 curves around 128 bits of security. λ denotes the security level, L the minimum of $r-1$ and $p-1$ 2-adicity, d the isogeny degree and α the smallest integer coprime to $r-1$.

u	p (bits)	r (bits)	λ $E(\mathbb{F}_p)$	λ $\mathbb{F}_{p^{12}}$	2-adicity L	d	α
`0x8508c00000000001`	377	253	126	126	47	2	11
`-0x7fb80fffffffffff`	377	252	126	126	45	2	5
`0x9b04000000000001`	379	254	127	126	51	2	7
`-0xfffbc3ffffffffff`	383	256	128	126	43	2	7
`-0xfff7c1ffffffffff`	383	256	128	126	42	2	7
`-0xffc3bfffffffffff`	383	256	128	126	47	2	7
`0x105a8000000000001`	383	257	128	126	52	2	7

Table 4.16: Seeds of SNARK-friendly inner BLS24 curves around 128 bits of security. λ denotes the security level, L the minimum of $r-1$ and $p-1$ 2-adicity, d the isogeny degree and α the smallest integer coprime to $r-1$.

u	p (bits)	r (bits)	λ $E(\mathbb{F}_p)$	λ $\mathbb{F}_{p^{24}}$	2-adicity L	d	α
`0x60300001`	305	245	122	158	22	2	7
`-0x950fffff`	311	250	125	159	22	2	7
`0x9f9c0001`	312	251	125	159	20	2	7
`-0xbfcfffff`	315	253	126	160	22	2	7
`-0xc90bffff`	315	254	126	160	20	2	13
`0xe19c0001`	317	255	127	160	20	2	17
`-0x10487ffff`	319	257	128	161	21	2	11

For outer curves, an additional property is

- Smallest $h_t^2 + 3h_y^2$ with low Hamming weight, allowing fast final exponentiation.

BLS12/BW6. The security of BLS12-384 and BLS12-448 is explained in [GS21], BLS12-448 being presented as a more conservative choice: it offers about 132 bits of security in $\mathbb{F}_{p^{12}}$ instead of 126 bits. Because a BLS12-448 would imply a much larger BW6-896, we concentrate on the BLS12 curves of 377 to 383 bits of Table 4.15. Given the above requirements, we short-list BLS12-377 with $u =$ `0x8508c00000000001` and BLS12-379 with $u =$ `0x9b04000000000001`. The former was proposed in [BCG+20] and used in [EHG20] and the latter was proposed in [EHG22], of a higher 2-adicity. Both have a $\mathrm{HW}(u) = 7$, $d = 2$, $\alpha \leq 7$ and tower fields can be constructed as $\mathbb{F}_p \xrightarrow{i^2+5} \mathbb{F}_{p^2} \xrightarrow{v^3-i} \mathbb{F}_{p^6} \xrightarrow{w^2-v} \mathbb{F}_{p^{12}}$.

Now, we construct outer BW6 curves to these inner BLS12 curves. For BLS12-377, we find BW6-761 to be optimal and refer the reader to [EHG20] for a more detailed study. For BLS12-379, we restrict the search to curves up to 768 bits and suggest the corresponding BW6-764 with $h_t = -23$, $h_y = 3$ and equation $Y^2 = X^3 + 1$ (and M-type twist $Y^2 = X^3 + 2$). Both BW6-761 and BW6-764 fall in the $t_{\mathrm{bw},3}$ case (Table 4.6).

BLS24/BW6. A BLS24 curve defined over a 320-bit prime field offers 128 bits of security on the curve thanks to a subgroup of prime order r of 256 bits, and offers around 160 bits in $\mathbb{F}_{p^{24}}$. Accordingly, we find the following SNARK-friendly inner BLS24 curves (Table 4.16). Given all the requirements, we choose BLS24-315 ($u =$ `-0xbfcfffff`) over

Table 4.17: BW6 outer curve parameters, where $y^2 = x^3 + b$.

outer curve	u	(h_t, h_y)	$t \bmod r, u$	b	$\mathbb{F}_{p^k}$ (bits)	est. DL in $\mathbb{F}_{p^k}$
BLS12-377-BW6-761	0x8508c00000000001	(13, 9)	0	-1	4566	126
BLS12-379-BW6-764	0x9b04000000000001	(-25, 3)	0	1	4584	126
BLS24-315-BW6-633	-0xbfcfffff	(- 7,-1)	0	4	3798	124
BLS24-315-BW6-672	-0xbfcfffff	(0x4dfff8,0)	0	-4	4032	128

$\mathbb{F}_p$ of 315 bits and with $\mathbb{F}_r$ of 253 bits. It has 2-adicity 22 and security level almost 128. The tower fields can be constructed as $\mathbb{F}_p \xrightarrow{i^2-13} \mathbb{F}_{p^2} \xrightarrow{v^2-i} \mathbb{F}_{p^4} \xrightarrow{w^2-v} \mathbb{F}_{p^8} \xrightarrow{c^3-w} \mathbb{F}_{p^{24}}$.

Now, we construct outer BW6 curves to BLS24-315. First, we search for less conservative curves over a field of up to 640 bits. We recommend the BW6-633 curve with $h_y = -7, h_y = -1$ and the equation $Y^2 = X^3 + 4$ (and M-type twist $Y^2 = X^3 + 8$). For more conservative curves offering 128 bits of security, we search for p_{bw} of exactly 672 bits. We recommend the BW6-672 curve with $h_t = 5111800, h_y = 0$ ($\text{HW}_{\text{2-NAF}}(h_t^2 + 2h_y^2) = 8$) and equation $Y^2 = X^3 - 4$ (D-type twist $Y^2 = X^3 - 4/3$). The former falls in the $t_{\text{bw},0}$ and the latter in the $t_{\text{bw},3}$ case.

Estimated complexity of a DL computation in $\text{GF}(p^k)$

This section recalls the results from [BD19, GS21, Gui20]. A BLS12 curve with r of about 256 bits has p of about 384 bits. In [GS21, Table 10] the estimated security in $\mathbb{F}_{p^{12}}$ for the BLS12-381 curve is 126 bits. Running the tool from [GS21], the paper [EHG20] shows that BLS12-377 in $\mathbb{F}_{p^{12}}$ has 125 bits of security, and BW6-761 has 126 bits of security in $\mathbb{F}_{p^6}$. With the same approach and the SageMath tool[2] from [GS21], our BLS12-379 curve has 125 bits in $\mathbb{F}_{p^{12}}$ and our BLS24-315 curve has 160 bits of security in $\mathbb{F}_{p^{24}}$.

We observe a notable difference between the BW6 outer curves of BLS12 and BLS24 because of the degree of the polynomial $p_{\text{bw}}(x)$. This polynomial is the key-ingredient of the Special (Tower) NFS [JP14, KB16]. However when its degree is too high, the general (Tower) NFS performs better, unless a tweak of $p_{\text{bw}}(x)$ is possible [Gui20]. This tweak divides by n the degree of $p_{bw}(x)$ while increasing its coefficients by at most u^{n-1}. It works only if either p_{bw} has an automorphism of degree n, hence the new polynomial has coefficients as small as the initial one, or the seed u is small enough. Here p_{bw} has no automorphism, and u is 32 bits long. We obtain a new $\tilde{p}_{\text{bw}}(x)$ of degree 10 and coefficients of 40 bits. The lowest estimate of DL cost with STNFS is 2^{132} with h of degree 6 for the 633-bit curve (cf. next paragraph for STNFS-security of MNT6 curves). The general TNFS works slightly better: with h of degree 2, and the Conjugation method (Conj), we obtain a DL cost estimate of 2^{124}. This is coherent with MNT-6 curve security estimates, where the same choice of parameters for TNFS apply [GMT20, Fig. 1]. To reach the 2^{128} cost, we increase p_{bw} up to 672 bits. We stress that the tool we use only gives an estimate, and recent progress are being made about TNFS [DGP21]. In case of underestimate of the tool, one can consider a 704-bit BLS24-BW6 curve.

For the Cocks-Pinch construction, the parameters do not have a polynomial form. For the embedding degree 8 we consider the TNFS-Conj algorithm with h of degree 2 according to [GMT20, Fig. 2]. We obtain 140 bits of security in $\mathbb{F}_{p^8}$ for the BLS24-315-CP8-632

[2]SageMath code available at `https://gitlab.inria.fr/tnfs-alpha/alpha`

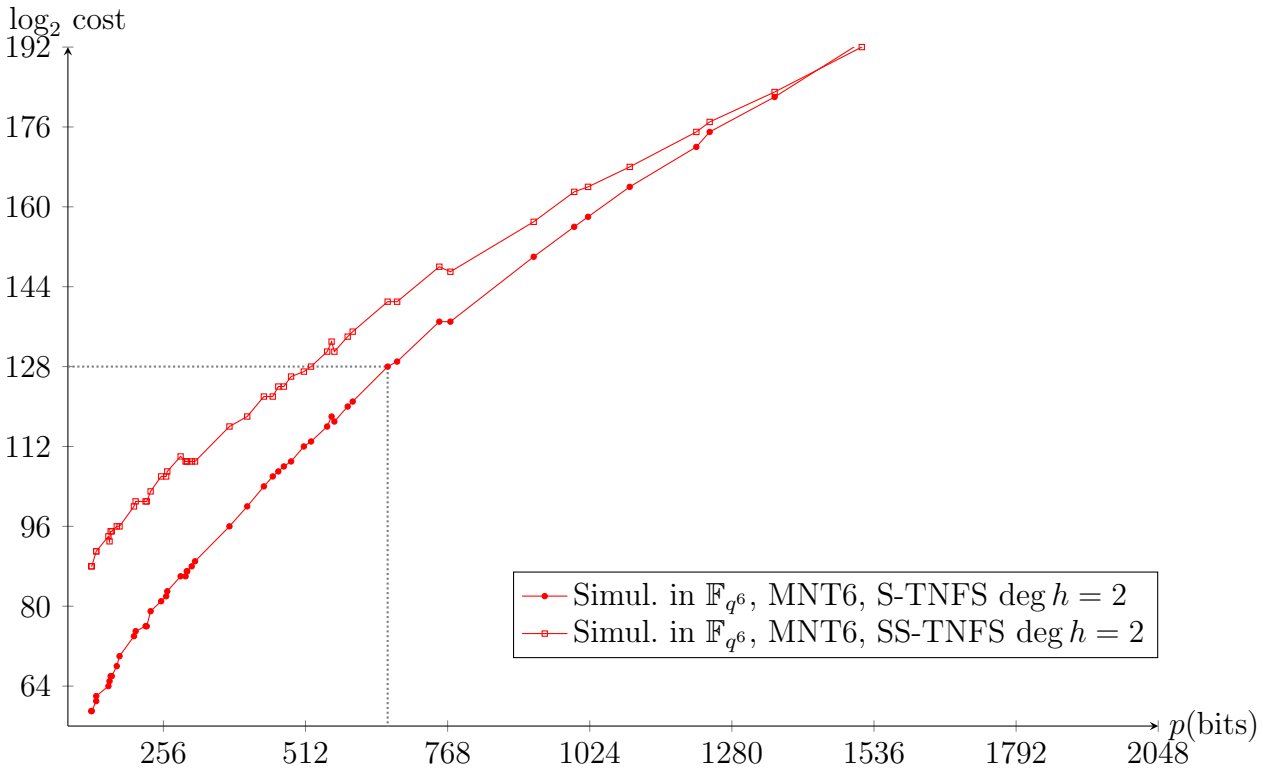


Figure 4.1: Estimated cost of DL computation with TNFS in GF(p^6).

curve. For the BLS24-315-CP12-630 curve we measure a DL cost of 166 bits in $\mathbb{F}_{p^{12}}$ with TNFS-Conj and h of degree 3 for the tower.

STNFS-security of MNT6 curves In [GMT20], Guillevic, Masson and Thomé estimated the cost of the Special-Tower Number Field Sieve algorithm (STNFS) and its variants for MNT6 curves (MNT curves of embedding degree 6) for curve parameters obtained from PBC library developed by Ben Lynn [Lyn13, Lyn07]. In [GS21], Guillevic and Singh refined the cost model. We reproduce in Fig. 4.1 the estimated cost of computing a discrete logarithm in GF(p^6) with the Tower NFS algorithm. There is a cross-over point at p of about 1536 bits from the Conjugation method of polynomial selection, to the generalisation made by Sarkar–Singh, both with the TNFS algorithm. The crossover point from TNFS to NFS is at much larger p. In conclusion, to ensure a 128-bit security level in a field GF(p^6), the prime p should be at least 672-bit long. If moreover a Special variant of NFS or TNFS is available because the prime p has a *special form*, the size requirement will be larger, but this is not the case for MNT parameters.

4.5.3 Golang library: implement the short-list curves

At `https://github.com/yelhousni/gnark-crypto` (`gnark-crypto` fork), we present an optimized implementation, with `x86-64` assembly code for the finite fields, of the short-listed curves: BLS12-377, BW6-761, BLS12-379, BW6-764, BLS24-315, BW6-633 and BW6-672 (Table 4.18). All curve implementations are written in Golang (tested with 1.16 and 1.17 versions) and benefit from $\mathbb{F}_p$ and $\mathbb{F}_r$ `x86-64` assembly accelerated arithmetic. Also, they benefit from $D = 3$ endomorphism-based optimizations (GLV and 2-dimensional GLS scalar multiplication, fast subgroup checks and cofactor clearing). For the pairing, we

Table 4.18: Short-listed curves.

curve, tower fields	equation	twist equation
BLS12-377, $\mathbb{F}_p \xrightarrow{i^2+5} \mathbb{F}_{p^2} \xrightarrow{v^3-i} \mathbb{F}_{p^6} \xrightarrow{w^2-v} \mathbb{F}_{p^{12}}$	$Y^2 = X^3 + 1$	$Y^2 = X^3 + 1/i$
BLS12-379, $\mathbb{F}_p \xrightarrow{i^2+5} \mathbb{F}_{p^2} \xrightarrow{v^3-i} \mathbb{F}_{p^6} \xrightarrow{w^2-v} \mathbb{F}_{p^{12}}$	$Y^2 = X^3 + 1$	$Y^2 = X^3 + 1/(5+i)$
BLS24-315, $\mathbb{F}_p \xrightarrow{i^2-13} \mathbb{F}_{p^2} \xrightarrow{v^2-i} \mathbb{F}_{p^4} \xrightarrow{w^2-v} \mathbb{F}_{p^8} \xrightarrow{c^3-w} \mathbb{F}_{p^{24}}$	$Y^2 = X^3 + 1$	$Y^2 = X^3 + 1/i$
BLS12-377-BW6-761, $\mathbb{F}_p \xrightarrow{i^3+4} \mathbb{F}_{p^3} \xrightarrow{v^2-i} \mathbb{F}_{p^6}$	$Y^2 = X^3 - 1$	$Y^2 = X^3 + 4$
BLS12-379-BW6-764, $\mathbb{F}_p \xrightarrow{i^3-2} \mathbb{F}_{p^3} \xrightarrow{v^2-i} \mathbb{F}_{p^6}$	$Y^2 = X^3 + 1$	$Y^2 = X^3 + 2$
BLS24-315-BW6-633, $\mathbb{F}_p \xrightarrow{i^3-2} \mathbb{F}_{p^3} \xrightarrow{v^2-i} \mathbb{F}_{p^6}$	$Y^2 = X^3 + 4$	$Y^2 = X^3 + 8$
BLS24-315-BW6-672, $\mathbb{F}_p \xrightarrow{i^3-3} \mathbb{F}_{p^3} \xrightarrow{v^2-i} \mathbb{F}_{p^6}$	$Y^2 = X^3 - 4$	$Y^2 = X^3 - 4/3$

Table 4.19: $\mathbb{G}_1$ and $\mathbb{G}_2$ scalar multiplication benchmarks.

curve	$\mathbb{G}_1$ scalar mul. (ns)	$\mathbb{G}_2$ scalar mul. (ns)
BLS12-377	77606	261607
BLS12-379	81090	272107
BLS24-315	65825	622044
BLS12-377-BW6-761	377360	377360
BLS12-379-BW6-764	390647	390647
BLS24-315-BW6-633	255600	255600
BLS24-315-BW6-672	300929	300929

follow optimizations from [ABLR14, Sco19, GS10, HHT20] and section 4.4.2. Our library is one of the fastest compared to other open-source libraries in Appendix A [EH].

4.5.4 Benchmarking

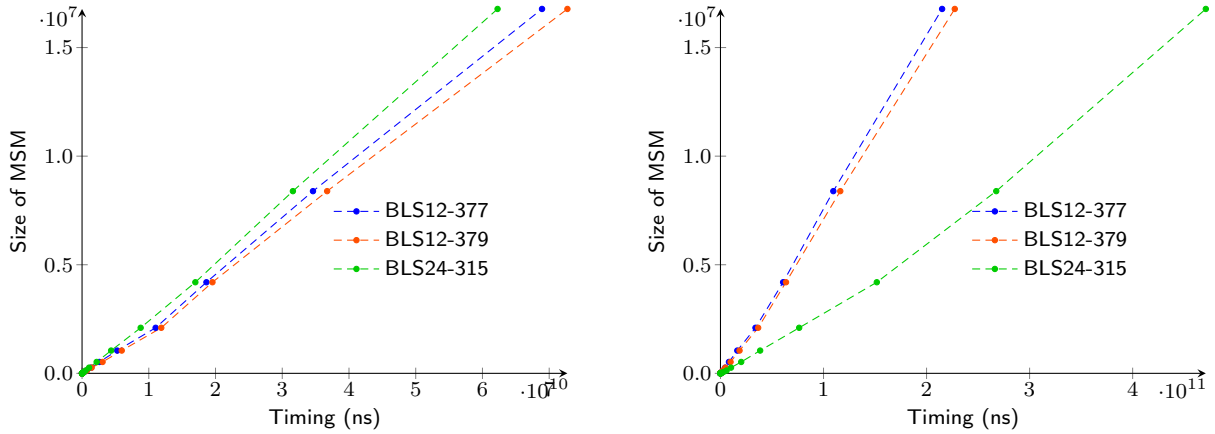
In this section, we benchmark our Golang implementation for all short-listed curves on two levels. First, independently from the context, we benchmark $\mathbb{G}_1$, $\mathbb{G}_2$ scalar multiplications (with GLV/GLS acceleration [GLV01, GLS09] and multi-scalar-multiplication (Bucket-list method [BDLO12, section 4]). Also, we benchmark the pairing computation (Miller loop, Final exponentiation and total pairing). Then, we benchmark the time to setup, prove and verify Groth16 and PLONK proofs of circuits with different number of constraints.

The first level benchmarks are run on an AWS z1d.large (3.4 GHz Intel Xeon) and the second level on a an AWS c5a.24xlarge (AMD EPYC 7R32). This allows to handle large proofs and to test different architectures. All with hyperthreading, turbo and frequency scaling disabled.

$\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ operations.

$\mathbb{G}_1$ coordinates for all short-listed curves are over $\mathbb{F}_p$ and use the $D = 3$ endomorphism to implement GLV [GLV01]. For $\mathbb{G}_2$, BW6 coordinates are over $\mathbb{F}_p$ as well and implement GLV ($D = 3$). For BLS12 and BLS24, the implementation uses 2-dimensional GLS [GLS09] over $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^4}$ respectively. Timings are reported in Tables 4.19 and 4.20. For multi-scalar-multiplication, we report timings in figures 4.2 and 4.3 for different sizes (2^5 to 2^{24} points).

On the one hand, we note that for inner curves BLS24-315 the arithmetic is the fastest



(a) $\mathbb{G}_1$-MSM (b) $\mathbb{G}_2$-MSM

Figure 4.2: MSM on $\mathbb{G}_1$ 4.2(a) and $\mathbb{G}_2$ 4.2(b) for short-listed inner curves.

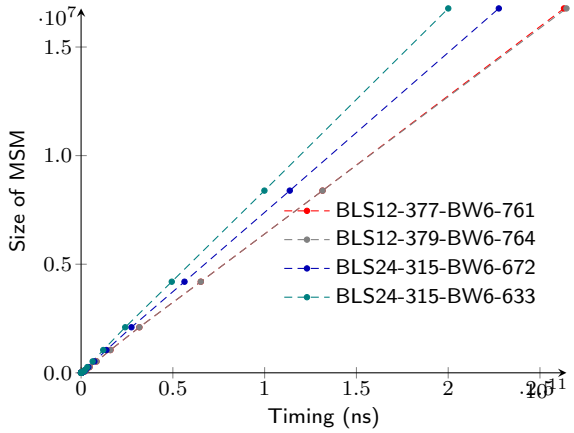


Figure 4.3: $\mathbb{G}_1/\mathbb{G}_2$-MSM on short-listed outer curves.

Table 4.20: Pairing computation benchmarks.

curve	Miller Loop (ns)	Final Exp. (ns)	Pairing (ns)
BLS12-377 opt. ate	377191	422157	799348
BLS12-379 opt. ate	383753	453687	837440
BLS24-315 opt. ate	435958	993500	1429458
BLS12-377-BW6-761 opt. ate (Eq. 4.15)	1613306	1099533	2712839
BLS12-377-BW6-761 opt. ate (Eq. 4.33)	1249860	1099533	2349393
BLS12-377-BW6-761 opt. Tate	1249860	1099533	2349393
BLS12-379-BW6-764 opt. ate (Eq. 4.15)	1548546	1057174	2605720
BLS24-315-BW6-633 opt. ate	918724	727918	1646642
BLS24-315-BW6-633 opt. Tate	809503	727918	1537421
BLS24-315-BW6-672 opt. ate	1073268	977436	2050704
BLS24-315-BW6-672 opt. Tate	973630	977436	1951066

Table 4.21: Cost of `Setup`, `Prove` and `Verify` algorithms for Groth16 and PLONK. m =number of wires, n =number of multiplications gates, a =number of additions gates and ℓ =number of public inputs. $\mathtt{M}_{\mathbb{G}}$ =multiplication in $\mathbb{G}$ and `P`=pairing.

	`Setup`	`Prove`	`Verify`
Groth16	$3n$ $\mathtt{M}_{\mathbb{G}_1}$, m $\mathtt{M}_{\mathbb{G}_2}$	$(3n+m-\ell)$ $\mathtt{M}_{\mathbb{G}_1}$, n $\mathtt{M}_{\mathbb{G}_2}$	3 `P`, ℓ $\mathtt{M}_{\mathbb{G}_1}$
PLONK (KZG)	$d_{\geq n+a}$ $\mathtt{M}_{\mathbb{G}_1}$, 1 $\mathtt{M}_{\mathbb{G}_2}$	$9(n+a)$ $\mathtt{M}_{\mathbb{G}_1}$	2 `P`, 18 $\mathtt{M}_{\mathbb{G}_1}$

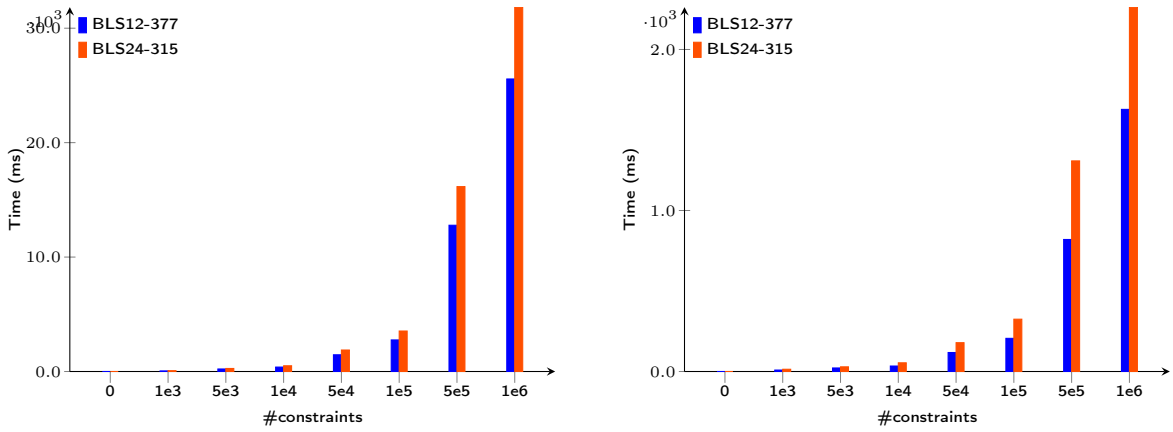
on $\mathbb{G}_1$, the slowest on $\mathbb{G}_2$ while still competitive on $\mathbb{G}_T$ (especially for multi-pairings when the final exponentiation is factored out). Thus, it is suitable for KZG-based SNARKs where only $\mathbb{G}_1$ operations and pairings accounts for the `Setup`, `Prove` and `Verify` algorithms. On the other hand, BLS12-377 presents the best trade-off on all operations making it suitable for Groth16 SNARK. For the less conservative choice of outer curve to BLS24-315, namely BW6-633, a pairing computation is almost as fast as on BLS24-315 and MSMs are the fastest on all outer curves given the small field size. For the conservative choice, namely BW6-672, operations on all three groups are reasonably fast and notably faster than on outer curves to BLS12 (BW6-761 and BW6-764).

Groth16 and PLONK schemes

Based on the analysis in the previous paragraph, here we discard BLS12-379/BW6-764 pair and choose to bench the BLS12-377/BW6-761 and BLS24-315/BW6-633/BW6-762 pairs of curves in the context of Groth16 and PLONK SNARKs. We choose a simple circuit (proof of exponentiation: $a^w := b$ (cf. Fig. 2.1)) to be able to control precisely the number of constraints. We bench the `Setup`, `Prove` and `Verify` algorithms for both Groth16 and PLONK schemes and report timings in figures 4.4, 4.5, 4.6, 4.7 and 4.8. The benchmark is run, this time, on an AWS c5a.24xlarge (AMD EPYC 7R32) to be able to test large circuits. In table 4.21 we recall the cost of SNARK algorithms in terms of preponderant groups operations.

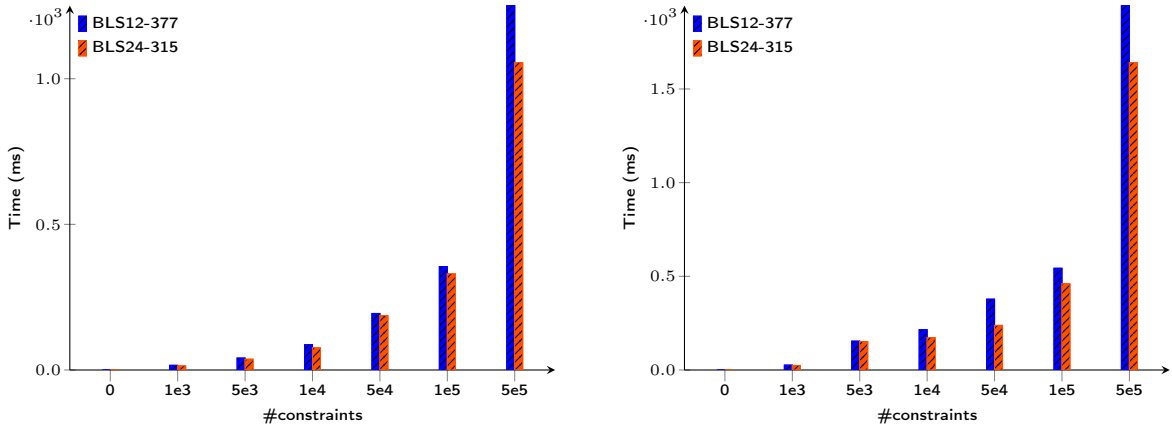
Remark 4.3. *The maximum number of constraints n_{max} a circuit can have is different per SNARK scheme and per curve. For Groth16, $n_{max} = 2^L$ and for PLONK $n_{max} = 2^{L-2}$ where L is the 2-adicity of the chosen curve.*

It is clear from figures 4.4, 4.5 and 4.8 that BLS12-377 is optimized to setup and prove Groth16 proofs while BLS24-315 is suitable to setup and prove PLONK proofs at the cost of acceptably slower verification time. For proof composition, we see from figures 4.6, 4.7 and 4.8 that the outer curves to BLS24-315, namely BW6-633 and BW6-672, are faster for all the SNARK algorithms for both Groth16 and PLONK. This confirms the recommendation of BLS24/BW6 pair of curves for KZG-based SNARK. We should also note that for applications where one would like to optimize the cost of generating and proving a proof of several proofs $\{\pi_i\}_{0\leq i\leq M}$ at the cost of slow generation of π_i (e.g. proof aggregation by light clients of off-chain generated proofs), one could use the BLS24/BW6 pair for Groth16.



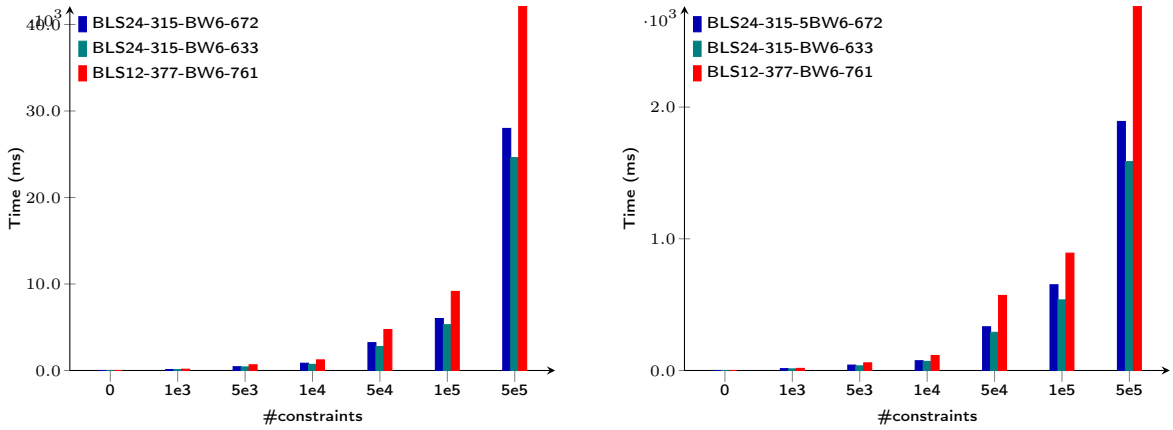
(a) Groth16 setup - inner curves (b) Groth16 prover - inner curves

Figure 4.4: Groth16 `Setup` (a) and `Prove` (b) times per number of constraints for inner curves.



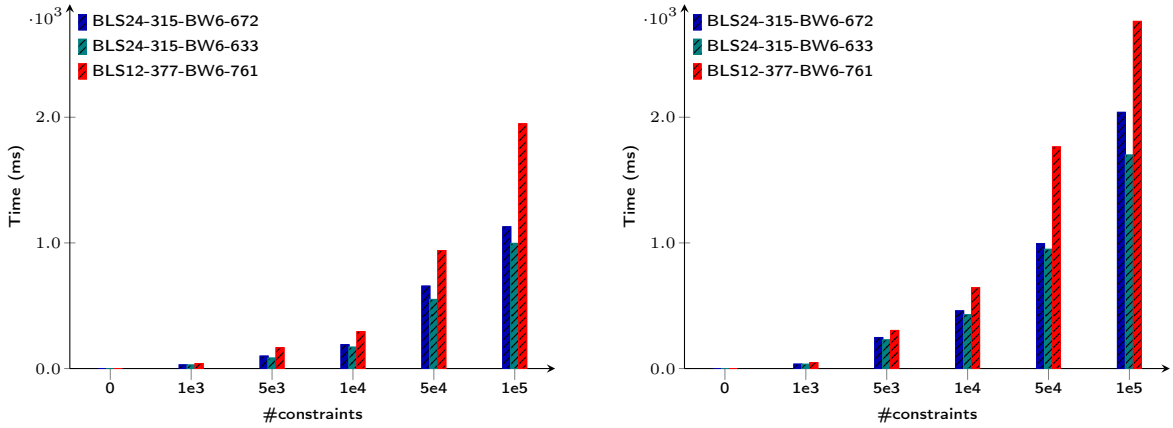
(a) PLONK setup - inner curves (b) PLONK prover - inner curves

Figure 4.5: PLONK `Setup` (a) and `Prove` (b) times per number of constraints for inner curves.



(a) Groth16 setup - outer curves (b) Groth16 prover - outer curves

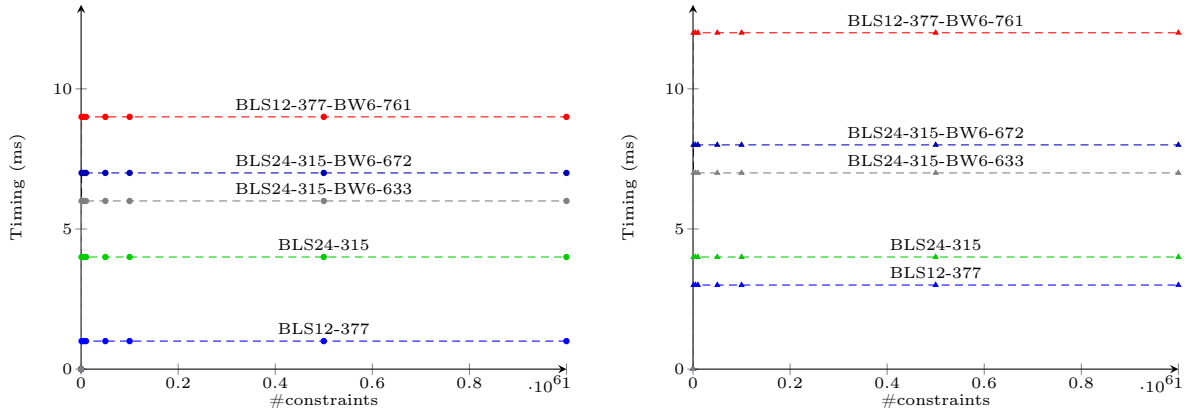
Figure 4.6: Groth16 `Setup` (a) and `Prove` (b) times per number of constraints for outer curves.



(a) PLONK setup - outer curves

(b) PLONK prover - outer curves

Figure 4.7: PLONK `Setup` (a) and `Prove` (b) times per number of constraints for outer curves.



(a) Groth16 verification

(b) PLONK verification

Figure 4.8: Groth16 (a) and PLONK (b) `Verify` times on short-listed curves.

Chapter

5

Multi-Scalar-Multiplication on SNARK-friendly curves

The bottleneck in the `Prove` algorithm of most of pairing-based SNARKs is the Multi-Scalar-Multiplication (MSM) algorithm. In this chapter we give an overview of a variant of the Pippenger MSM algorithm together with a set of optimizations we proposed. This chapter, in part, is a reprint of the material as it appears in our submitted work [HB22]. We implemented this work in `gnark-crypto` as part of our submission to the ZPrize competition (`https://www.zprize.io/`). We participated in the open division "Accelerating MSM on Mobile". This prize focus on minimizing latency of computing MSM in native mobile applications over the BLS12-377 curve. We achieved an 78% speedup over the ZPrize baseline implementation, which allowed us to win the first prize.

Outline. Given a set of n elements $G_1, \cdots, G_n$ (bases) in $\mathbb{G}$ a cyclic group whose order $\#\mathbb{G}$ has b-bit and a set of n integers $a_1, \cdots, a_n$ (scalars) between 0 and $\#\mathbb{G}$, the goal is to compute efficiently the group element $[a_1]G_1 + \cdots + [a_n]G_n$. In SNARK applications, we are interested in large instances of variable-base MSMs ($n = 10^7, 10^8, 10^9$) — with random bases and random scalars — over the groups $\mathbb{G}_1$ and $\mathbb{G}_2$.

The naive algorithm uses a double-and-add strategy to compute each $[a_i]G_i$ then adds them all up, costing on average $3/2 \cdot b \cdot n$ group operations ($+$). There are several algorithms that optimize the total number of group operations as a function of n such as Strauss [Str64], Bos–Coster [dR95, Sec. 4] and Pippenger [Pip76] algorithms. For large instances of a variable-base MSM, the fastest approach is a variant of Pippenger's algorithm [BDLO12, Section 4]. For simplicity, we call it the bucket-list method.

5.1 Pippenger variant: the bucket-list method

The high-level strategy is in three steps:

- Step 1: reduce the b-bit MSM to several c-bit MSMs for some fixed $c \leq b$
- Step 2: solve each c-bit MSM efficiently
- Step 3: combine the c-bit MSMs into the final b-bit MSM

Step 1: reduce the b-bit MSM to several c-bit MSMs

1. Choose a window $c \leq b$
2. Write each scalar $a_1, \cdots, a_n$ in binary form and partition each into c-bit parts

$$a_i = \underbrace{(a_{i,1}, a_{i,2}, \cdots, \underbrace{a_{i,b/c}}_{c\text{-bit}})_2}_{b\text{-bit}}$$

3. Deduce b/c instances of c-bit MSMs from the partitioned scalars

$$\begin{aligned} T_1 &= [a_{1,1}]G_1 + \cdots + [a_{n,1}]G_n \\ &\vdots \\ T_j &= [a_{1,j}]G_1 + \cdots + [a_{n,j}]G_n \\ &\vdots \\ T_{b/c} &= [a_{1,b/c}]G_1 + \cdots + [a_{n,b/c}]G_n \end{aligned}$$

Cost of Step 1 is negligible.

Step 3: combine the c-bit MSMs into the final b-bit MSM

Algorithm 5.1: Step 3: combine the c-bit MSMs into the final b-bit MSM
Input: The set $\{T_i\}_{i=1}^{b/c}$ output of Step 2
Output: $T = [a_1]G_1 + \cdots + [a_n]G_n$

```
T ← T_1;
for i from 2 to b/c:
    T ← [2^c]T;                    // Double c times
    T ← T + T_i;                   // Add
return T;
```

Cost of Step 3: $(b/c - 1)(c + 1) = b - c + b/c - 1$ group operations.

Step 2: solve each c-bit MSM T_j efficiently

1. For each T_j, accumulate the bases G_i inside buckets
   Each element $a_{i,j}$ is in the set $\{0, 1, 2, \cdots 2^c - 1\}$. We initialize $2^c - 1$ empty buckets (with points at infinity) and accumulate the bases G_i from each T_j inside the bucket corresponding to the scalar $a_{i,j}$.

			G_k		
			$+$		
			$\vdots$		
			$+$		
	G_{2^c-k}		G_{23}		
	$+$		$+$		
	G_7	G_{15}	G_{19}		$G_{2^c-k'}$
	$+$	$+$	$+$	$\vdots$	$+$
	G_4	G_3	G_{18}		G_1
buckets:	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$	$\cdots$	$\boxed{2^c-1}$
sum:	S_1	S_2	S_3	$\cdots$	S_{2^c-1}

Cost: $n - (2^c - 1) = n - 2^c + 1$ group operations.

2. Combine the buckets to compute T_j
   This step is also a c-bit MSM of size $2^c - 1$ but this time the scalars are ordered and known in advance $S_1 + [2]S_2 + \cdots + [2^c - 1]S_{2^c-1}$, thus we can compute this instance efficiently as follows

	S_{2^c-1}											
$+$	S_{2^c-1}	$+$	S_{2^c-2}									
	$\vdots$											
$+$	S_{2^c-1}	$+$	S_{2^c-2}	$+$	$\cdots$	$+$	S_3	$+$	S_2			
$+$	S_{2^c-1}	$+$	S_{2^c-2}	$+$	$\cdots$	$+$	S_3	$+$	S_2	$+$	S_1	
$[2^c-1]S_{2^c-1}$		$+$	$[2^c-2]S_{2^c-2}$	$+$	$\cdots$	$+$	$[3]S_3$	$+$	$[2]S_2$	$+$	S_1	

Cost: $2(2^c - 2) = 2^{c+1} - 4$ group operations.

Cost of Step 2: $n - 2^c + 1 + 2^{c+1} - 4 = n + 2^c - 3$ group operations.

Combining Steps 1, 2 and 3, the expected overall cost of the bucket-list method is

Total cost: $\frac{b}{c}(n + 2^c - 3) + (b - c + b/c - 1) \approx \frac{b}{c}(n + 2^c)$ group operations.

Remark 5.1 (On choosing c)**.** *The theoretical minimum occurs at $c \approx \log n$ and the asymptotic scaling looks like $(b\frac{n}{\log n})$. However, in practice, empirical choices of c yield a better performance because the memory usage scales with 2^c and there are fewer edge cases if c divides b. For example, with $n = 10^7$ and $b = 256$, we observed a peak performance at $c = 16$ instead of $c = \log n \approx 23$.*

5.2 Our optimizations

Parallelism

Since each c-bit MSM is independent of the rest, we can compute each (Step 2) on a separate core. This makes full use of up to b/c cores but increases memory usage as each core needs $2^c - 1$ buckets (points). If more than b/c cores are available, further parallelism does not help much because m MSM instances of size n/m cost more than 1 MSM instance of size n.

Precomputation

When the bases $G_1, \cdots, G_n$ are known in advance, we can use a smooth trade-off between precomputed storage vs. run time. For each base G_i, choose k as big as the storage allows and precompute k points $[2^c{-}k]G, \cdots, [2^c - 1]G$ and use the bucket-method only for the first $2^c - 1{-}k$ buckets instead of $2^c - 1$. The total cost becomes $\approx \frac{b}{c}(n + 2^c{-}k)$. However, large MSM instances already use most available memory. For example, when $n = 10^8$ our implementation needs 58GB to store enough BLS12-377 curve points to produce a Groth16 proof. Hence, the precomputation approach yield negligible improvement in our case.

Algebraic structure

Since the bases $G_1, \cdots, G_n$ are points in $\mathbb{G}_1$ or $\mathbb{G}_2$, we can use the algebraic structure of elliptic curves to further optimize the bucket-list method.

Non-Adjacent-Form (NAF). Given a point $G_i = (x, y) \in \mathbb{G}_1$ (or $\mathbb{G}_2$), on a Weierstrass curve for instance, the negative $-G_i$ is $(x, -y)$. This observation is well known to speed up the scalar multiplication $[s]G_i$ by encoding the scalar s in a signed binary form $\{-1, 0, 1\}$ (later called 2-NAF — the first usage might go back to 1989 [MO90]). However, this does not help in the bucket-list method because the cost increases with the number of possible scalars regardless of their encodings. For a c-bit scalar, we always need $2^c - 1$ buckets. That is said, we can use the 2-NAF decomposition differently. Instead of writing the c-bit scalars in the set $\{0, \cdots, 2^c - 1\}$, we write them in the signed set $\{-2^{c-1}, \cdots, 2^{c-1} - 1\}$ (cf. Alg. 5.2). If a scalar $a_{i,j}$ is strictly positive we add G_i to the bucket $S_{(a_{i,j})_2}$ as usual, and if $a_{i,j}$ is strictly negative we add $-G_i$ to the bucket $S_{|(a_{i,j})_2|}$. This way we reduce the number of buckets by half.

Total cost: $\approx \frac{b}{c}(n + 2^{c-1})$ group operations.

The signed-digit decomposition cost is negligible but it works only if the bitsize of $\#\mathbb{G}_1$ (and $\#\mathbb{G}_2$) is strictly bigger than b. We use the spare bits to avoid the overflow. This observation was taken into account at the curve design level (cf. Sec. 4.5.2).

Curve forms and coordinate systems. To minimize the overall cost of storage but also run time, we store the bases G_i in affine coordinates. This way we only need the tuples (x_i, y_i) for storage (although we can batch-compress these following [Kos21]) and we can make use of mixed addition with a different coordinate systems.

Algorithm 5.2: Signed-digit decomposition
Input: $(a_0, \cdots, a_{b/c-1}) \in \{0, \cdots, 2^c - 1\}$
Output: $(a'_0, \cdots, a'_{b/c-1}) \in \{-2^{c-1}, \cdots, 2^{c-1} - 1\}$

for i *from* 0 *to* $b/c - 1$**:**
  if $a_i \geq 2^{c-1}$**:**
    assert $i \neq b/c - 1$; // No overflow for the final digit
    $a'_i \leftarrow a_i - 2^c$; // Force this digit into $\{-2^{c-1}, \cdots, 2^{c-1} - 1\}$
    $a_{i+1} \leftarrow a_{i+1} + 1$; // Lend 2^c to the next digit
  else:
    $a'_i \leftarrow a_i$
return $(a'_0, \cdots, a'_{b/c-1})$;

The overall cost of the bucket-list method is $\frac{b}{c}(n + 2^{c-1}) + (b - c - b/c - 1)$ group operations. This can be broken down explicitly to:

- Mixed additions: to accumulate G_i in the c-bit MSM buckets with cost $\frac{b}{c}(n - 2^{c-1} + 1)$
- Additions: to combine the bucket sums with cost $\frac{b}{c}(2^c - 3)$
- Additions and doublings: to combine the c-bit MSMs into the b-bit MSM with cost $b - c + b/c - 1$
  - $b/c - 1$ additions and
  - $b - c$ doublings

For large MSM instances, the dominating cost is in the mixed additions as it scales with n. For this, we use extended Jacobian coordinates $\{X, Y, ZZ, ZZZ\}$ ($x = X/ZZ, y = Y/ZZZ, ZZ^3 = ZZZ^2$) trading-off memory for run time compared to the usual Jacobian coordinates $\{X, Y, Z\}$ ($x = X/Z^2, y = Y/Z^3$) (cf. Table 5.1).

Coordinate systems	Mixed addition	Addition	Doubling
Jacobian	$7\mathbf{m} + 4\mathbf{s}$	$11\mathbf{m} + 5\mathbf{s}$	$2\mathbf{m} + 5\mathbf{s}$
Extended Jacobian	$8\mathbf{m} + 2\mathbf{s}$	$12\mathbf{m} + 2\mathbf{s}$	$6\mathbf{m} + 4\mathbf{s}$

Table 5.1: Cost of arithmetic in Jacobian and extended Jacobian coordinate systems. **m**=Multiplication and **s**=Squaring in the field.

Remark 5.2. *In [GW20], the authors suggest to use affine coordinates for batch addition. That is, they only compute the numerators in the affine addition, accumulate the denominators and then batch-invert them using the Montgomery trick [Mon87]. An affine addition costs* $3\mathbf{m} + 1\mathbf{i}$ *(*$\mathbf{i}$ *being a field inversion). For a single addition this is not worth it as* $1\mathbf{i} > 7\mathbf{m}$ *(*$= 10\mathbf{m} - 3\mathbf{m}$*). If we accumulate* L *points and batch-add them with cost* $3L\mathbf{m} + L\mathbf{i} = 6L\mathbf{m} + 1\mathbf{i}$ *(the Montgomery trick costing* $L\mathbf{i} = 3L\mathbf{m} + 1\mathbf{i}$*), this might be worth it. Assuming* $I = C\mathbf{m}$*, there might be an improvement if we accumulate a number of points* $L > C/4$*. However, we did not observe a significant improvement in our implementation in* `gnark-crypto` *compared to the extended Jacobian approach. This is mainly because* C

is large due the optimized finite field arithmetic in `gnark-crypto` *[gna20]. This means L should be large requiring more memory and furthermore one needs to implement specialized batch-addition functions in affine coordinates.*

We work over fields of large prime characteristic ($\neq 2, 3$), so the elliptic curves in question have always a short Weierstrass (SW) form $y^2 = x^3 + ax + b$. Over this form, the fastest mixed addition is achieved using extended Jacobian coordinates. However, there are other forms that enable even faster mixed additions (cf. Table 5.2).

Form	Coordinates system	Equation	Mixed addition cost
short Weierstrass	extended Jacobian	$y^2 = x^3 + ax + b$	10**m**
Jacobi quartics	$XXYZZ$, doubling-oriented $XXYZZ$, $XXYZZR$, doubling-oriented $XXYZZR$	$y^2 = x^4 + 2ax^2 + 1$	9**m**
Edwards	projective, inverted	$x^2 + y^2 = c^2(1 + dx^2y^2)$	9**m**
twisted Edwards	extended $(XYZT)$ $x = X/Z, y = Y/Z, x \cdot y = T/Z$	$ax^2 + y^2 = 1 + dx^2y^2$	8**m** (dedicated) 9**m** (unified)
twisted Edwards	extended $(XYZT)$ $x = X/Z, y = Y/Z, x \cdot y = T/Z$	$-x^2 + y^2 = 1 + dx^2y^2$ $(a = -1)$	7**m** (dedicated) 8**m** (unified)

Table 5.2: Cost of mixed addition in different elliptic curve forms and coordinate systems assuming 1**m** = 1**s**. Formulas and references from [BL22].

It appears that a twisted Edwards (tEd) form is appealing for the bucket-list method since it has the lowest cost for the mixed addition in extended coordinates. Furthermore, the arithmetic on this form is *complete*, i.e. the addition formulas are defined for all inputs. This improves the run time by eliminating the need of branching in case of adding the neutral element or doubling compared to a SW form. We show in Lemma 5.1 that all inner BLS curves (cf. Sec. 4.4.1) admit a tED form.

Lemma 5.1. *All inner BLS curves (cf. Sec. 4.4.1) admit a twisted Edwards form* $ay^2+x^2 = 1 + dx^2y^2$ *with* $a = 2\sqrt{3} - 3$ *and* $d = -2\sqrt{3} - 3$ *over* $\mathbb{F}_p$. *If further* $-a$ *is a square, the equation becomes* $-x^2 + y^2 = 1 + d'x^2y^2$ *with* $d' = 7 + 4\sqrt{3} \in \mathbb{F}_p$.

Proof. Proposition 4.1 shows that all inner BLS curves are of the form $W_{0,1} : y^2 = x^3 + 1$. The following map

$$\begin{aligned} W_{0,1} &\to E_{a,d} \\ (x, y) &\mapsto \left(\frac{x+1}{y}, \frac{x+1-\sqrt{3}}{x+1+\sqrt{3}}\right) \end{aligned}$$

defines the curve $E_{a,d} : ay^2 + x^2 = 1 + dx^2y^2$ with $a = 2\sqrt{3} - 3$ and $d = -2\sqrt{3} - 3$. The inverse map is

$$\begin{aligned} E_{a,d} &\to W_{0,1} \\ (x, y) &\mapsto \left(\frac{(1+y)\sqrt{3}}{1-y} - 1, \frac{(1+y)\sqrt{3}}{(1-y)x}\right) \end{aligned}$$

If $-a$ is a square in $\mathbb{F}_p$, the map $(x, y) \mapsto (x/\sqrt{-a}, y)$ defines from $E_{a,d}$ the curve $E_{-1,d'}$ of equation $-x^2 + y^2 = 1 + d'x^2y^2$ with $d' = -d/a = (2\sqrt{3}+3)/(2\sqrt{3}-3) = 7 + 4\sqrt{3}$.

These maps work only if $\sqrt{3}$ is defined in $\mathbb{F}_p$, that is 3 is a quadratic residue. This is always the case in $\mathbb{F}_p$ on which an inner BLS curve is defined. Let $\left(\frac{3}{p}\right)$ be $3^{\frac{p-1}{2}} \mod p$, the Legendre symbol. The quadratic reciprocity theorem tells us that $\left(\frac{3}{p}\right)\left(\frac{p}{3}\right) = (-1)^{\frac{p-1}{2}}$. We have $p \equiv 1 \mod 4$ from the 2-adicity condition, so $\left(\frac{3}{p}\right) = \left(\frac{p}{3}\right)$. Now $\left(\frac{p}{3}\right) \equiv p \mod 3$ which is always equal to 1 for all BLS curves ($x \equiv 1 \mod 3$ and $x-1 \mid p-1$). More generally one can prove that when $p = 2$ or $p \equiv 1$ or $11 \mod 12$ then 3 is a quadratic residue in $\mathbb{F}_p$. For inner BLS, we have $p \equiv 1 \mod 3 \cdot 2^L$ with $L \gg 2$. □

For the arithmetic, we use the formulas in [HWCD08] alongside some optimizations. We take the example of BLS12-377 for which $a = -1$:

- To combine the c-bit MSMs into a b-bit MSM we use unified additions [HWCD08, Sec. 3.1] ($9\mathbf{m}$) and dedicated doublings [HWCD08, Sec. 3.3] ($4\mathbf{m} + 4\mathbf{s}$).
- To combine the bucket sums we use unified additions ($9\mathbf{m}$) to keep track of the running sum and unified re-additions ($8\mathbf{m}$) to keep track of the total sum. We save $1\mathbf{m}$ by caching the multiplication by $2d'$ from the running sum.
- To accumulate the G_i in the c-bit MSM we use unified re-additions with some precomputations. Instead of storing G_i in affine coordinates we store them in a custom coordinates system (X, Y, T) where $y - x = X$, $y + x = Y$ and $2d' \cdot x \cdot y = T$. This saves $1\mathbf{m}$ and $2\mathbf{a}$ (additions) at each accumulation of G_i.

We note that although the dedicated addition (resp. the dedicated mixed addition) in [HWCD08, Sec. 3.2] saves the multiplication by $2d'$, it costs $4\mathbf{m}$ (resp. $2\mathbf{m}$) to check the operands equality: $X_1Z_2 = X_2Z1$ and $Y_1Z_2 = Y_2Z1$ (resp. $X_1 = X_2Z1$ and $Y_1 = Y_2Z1$). This cost offset makes both the dedicated (mixed) addition and the dedicated doubling slower than the unified (mixed) addition in the MSM case. We also note that the conversion of all the G_i points given on a SW curve with affine coordinates to points on a tEd curve (also with $a = -1$) with the custom coordinates (X, Y, T) is a one-time computation dominated by a single inverse using the Montgomery batch trick. In proof systems, since the G_i are points from the proving key σ_p, this computation can be part of the `Setup` algorithm and do not impact the `Prove` algorithm. If the `Setup` ceremony is yet to be conducted, it can be performed directly with points in the twisted Edwards form.

Our implementation in `gnark-crypto` shows that an MSM instance of size 2^{16} on the BLS12-377 curve is 30% faster when the G_i points are given on a tEd curve with the custom coordinates compared to the Jacobian-extended-based version which takes points in affine coordinates on a SW curve.

Endomorphisms. All the curves studied in this thesis have a j-invariant 0 (or 1728 for CP8). These curves are equipped with fast endomorphisms (cf. Sec. 1.2). On $\mathbb{G}_1$, the endomorphism can help transform a b-bit MSM of size n to a $b/2$-bit MSM of size $2n$

$$[a_1]G_1 + \cdots + [a_n]G_n = [a_{1,1}]G_1 + [a_{1,2}]\phi(G_1) + \cdots + [a_{n,1}]G_n + [a_{n,2}]\phi(G_n)$$

Intuitively, it would seem that these two MSM instances would have similar complexity but the larger MSM uses less buckets $b/2c \cdot (2n + 2^{c-1}) = b/c \cdot (n + 2^{c-2})$. The cost of computing $G_i \mapsto \phi(G_i)$ is negligible but the cost of scalar decomposition $a_i \mapsto (a_{i,1}, a_{i,2})$ should be taken into account as it involves a Babai rounding (division). However, since the bucket-list method is not a constant-time algorithm anyway the cost of the scalar decomposition can be reduced by replacing the division by a right-shift and checking the sub-scalars bounds at runtime. In fact, we can precompute $e = 2^m \cdot v_i/d$ (with v_i the short basis vector, d the basis determinant and $m > \log_2(d)$) and then at runtime compute $a_i \cdot e/2^m$ (right-shift). This increases the bounds on sub-scalars $(a_{i,1}, a_{i,2})$ by 1 which we check at runtime before increasing the size of b (not constant-time). We choose m to be a machine word twice bigger than $\log_2(d)$ so that we rarely increase the size of b. This is similar to the idea presented in [CL15] for scalar multiplication.

On $\mathbb{G}_2$, higher-dimensional decompositions are available. For example on BLS12 and CP8, we can have a 4-dimensional decomposition and on BLS24 a 8-dimensional one. On BW6, only a 2-dimensional decomposition is available as in $\mathbb{G}_1$.

Note: when accumulating the bases in a bucket it might happen that we add G_i and $\phi(G_i)$ which costs less than an addition (e.g. $G_i + \phi(G_i) = -\phi^2(G_i)$ for $j = 0$ curves). However, empirically, for large MSM instances it is cheaper to do the plain addition than to check for these cases.

Open question: for Groth16, the same scalars a_i are used for both $\mathbb{G}_1$ and $\mathbb{G}_2$ MSMs. Is it possible to mutualize a maximum of computations between these two instances? It seems that moving to a type-2 pairing would allow to deduce the $\mathbb{G}_1$ instance from the $\mathbb{G}_2$ one using an efficient homomorphism (the trace map) over the resulting single point. However, $\mathbb{G}_2$ computations would be done on the much slower full extension $\mathbb{F}_{p^k}$. The pairing, needed for proof verification, would also be moderately slower because of the anti-trace map.

5.3 Implementation

Submissions to the ZPrize "Accelerating MSM on Mobile" division must run on Android 12 (API level 32) and are tested on the Samsung Galaxy A13 5G (Model SM-A136ULGDXAA with SoC MediaTek Dimensity 700 (MT6833)). The MSM must be an instance of 2^{16} $\mathbb{G}_1$-points on the BLS12-377 curve. The baseline is the `arkworks` [ac22] MSM implementation in Rust (the bucket-list method). Submissions must beat this baseline by at least 10% in order to be eligible for the prize. We achieved a speedup of 78% (cf. Table 5.3), which allowed us to win the first prize.

`https://github.com/gbotrel/zprize-mobile-harness`

The speedup comes from the optimized finite field arithmetic in `gnark-crypto` compared to `arkworks` and the algorithmic optimizations discussed in this chapter. However, the large gap cannot be justified by these facts only. The target device SoC can run 32-bit and 64-bit instruction sets. However, the stock firmware runs a 32-bit ARM architecture (armv7) on which the baseline implementation is benchmarked by the ZPrize judges. For the sake of the competition, we performed a static build targeting a 64-bit ARM architecture (arm64), which allowed us without a complicated build process to run the 64-bit code on the target device. We also added $\sim$ 40 lines of ARM assembly for a small function

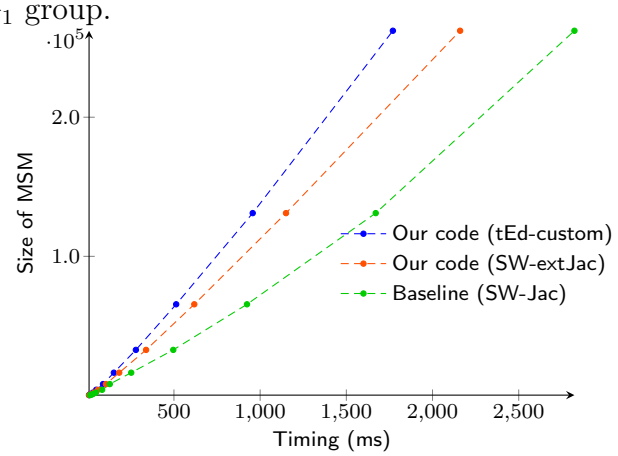
Implementation	Timing	Curve form and coordinates system	Parallelism?	Precom-putation?	2-NAF buckets?	Endo-morphism?
Baseline	2309 ms	SW Jacobian (X, Y, Z)	✓	✗	✗	✗
Submission	509 ms	tEd $(a = -1)$ Custom (X, Y, T)	✓	✗	✓	✗

Table 5.3: Comparison of the ZPrize baseline and the submission MSM instances of 2^{16} $\mathbb{G}_1$-points on the BLS12-377 curve.

in $\mathbb{F}_p$ (`Butterfly(a, b)` $\rightarrow$ `a = a + b; b = a - b`). The performance impact was 5%, as it speeds up a bit the unified (mixed) addition in the tEd form.

For the sake of this chapter, and for a fair comparison, we perform the same architecture hack on the baseline implementation. We report in Figure 5.1 a comparison of our code to the baseline. We report timings of several MSM instances of different sizes and with different curve parameterizations (SW in extended Jacobians vs. tEd $(a = -1)$ in custom/extended coordinates).

Figure 5.1: Comparison of our MSM code and the ZPrize baseline for different instances on the BLS12-377 $\mathbb{G}_1$ group.



For the ZPrize MSM instance of size 2^{16} the speed up is 45% with the submitted tEd-custom version and 33% with the more generic SW-extJac version. For different sizes ranging from 2^8 to 2^{18} the speed up is 40-47% with the tEd-custom version and 20-35% with SW-extJac.

III

Elliptic curves and pairings in SNARKs

Chapter

6

Elliptic curve arithmetic in rank-1 constraint systems

In this chapter, we introduce the notion of R1CS as a computation model for SNARKs. We particularly investigate elliptic curve arithmetic as a computation to be proven. This needs tailored constructions of elliptic curves and optimizations that are specific to the R1CS model. This chapter, in part, is a reprint of the material as it appears in our published work [AEHG22].

6.1 Rank-1 constraint system

The first step in SNARK-proving an arbitrary computation is to *arithmetize* it, that is to reduce the computation satisfiability to an intermediate representation satisfiability. Many problems in cryptography can be expressed as the task of computing some polynomials. Arithmetic circuits are the most standard model for studying the complexity of such computations.

Arithmetic circuits

An arithmetic circuit $\mathcal{A}$ over the field $\mathbb{F}$ and the set of variables $X = \{x_0, \ldots, x_n\}$ is a directed acyclic graph such that the vertices of $\mathcal{A}$ are called gates, while the edges are called wires. Arithmetic circuits of interest to many SNARKs and most applicable to this work are those whose gates have two incoming wires and one outcoming wire (cf. Fig. 6.1 for an example).

R1CS

SNARKs, such as [Gro16], express these arithmetic circuits as a set of quadratic constraints called Rank-1 Constraint System (R1CS). It consists of two subsets of constraints:

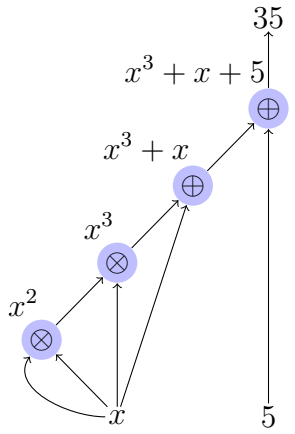


Figure 6.1: Arithmetic circuit encoding the computation $x^3 + x + 5 = 35$ for which the (secret) solution is $x = 3$.

multiplication gates and linear constraints in terms of the circuit variables. There are two kinds of variables in the constraint system: the input secrets v_i and the internal inputs and outputs of the multiplication gates. Each multiplication gate takes two inputs and outputs their product. The relation for n gates is represented as

$$\vec{a_L} \circ \vec{a_R} = \vec{a_O},$$

where $\vec{a_L} = (a_{L,1}, \ldots, a_{L,n})$ is the vector of the first (left) input to each gate, $\vec{a_R} = (a_{R,1}, \ldots, a_{R,n})$ the vector of the second (right) input to each gate and $\vec{a_O} = (a_{L,1} \cdot a_{R,1}, \ldots, a_{L,n} \cdot a_{R,n}) = (a_{O,1}, \ldots, a_{O,n})$ the vector of the output, also known as the Hadamard relation. Linear constraints are expressed using a vector of equations that use linear combinations of the variables as

$$\vec{W_L}\vec{a_L} + \vec{W_R}\vec{a_R} + \vec{W_O}\vec{a_O} = \vec{W_v}\vec{v} + \vec{c},$$

where $\vec{W_L}$, $\vec{W_R}$ and $\vec{W_O}$ are weight vectors applied to the respective inputs and outputs of the internal variables, $\vec{W_v}$ are weight vectors applied to the inputs variables $\vec{v}$ and $\vec{c}$ is a vector of constant terms used in the linear constraints.

SNARK-friendly computations

Many SNARK constructions model computations to be proven as R1CS circuits where the variables are in a field $\mathbb{F}$ where the discrete logarithm is hard. In pairing-based SNARKs the field is chosen to be $\mathbb{F}_r$, where r is the prime subgroup order on the curve. The number of these variables and particularly the number of multiplication gate variables determines the prover runtime complexity. For example, Groth16 prover complexity is dominated by the multi-scalar-multiplications (in $\mathbb{G}_1$ and $\mathbb{G}_2$) of size n (the number of multiplication gates). With this in mind, additions and constant-scalar multiplications in $\mathbb{F}_r$, which are usually expensive in hardware, are essentially free. While more traditional hardware-friendly computations (e.g. XORing 32-bit numbers) are far more costly in R1CS. The following two observations, noted in earlier works [KZM+15], are the key to lower the number of multiplication gates of a SNARK circuit:

- Additions and multiplications by constants in $\mathbb{F}_r$ are free;

- The verification can be sometimes simpler than the forward computation. The SNARK circuits do not always have to compute the result, but can instead represent a verification algorithm. For example a multiplicative inversion circuit ($1/x \stackrel{?}{=} y$) does not have to encode the computation of the inversion ($1/x$) but can instead consist of a single multiplication constraint ($x \cdot y$) on the value provided (precomputed) by the prover (y) and checks the equality ($x \cdot y \stackrel{?}{=} 1$).

This is basically a computation model where inversions cost (almost) as much as multiplications.

6.2 Elliptic curves inside a SNARK

While SNARKs allow proving general-purpose computations, in many applications these computations revolve around proving some cryptographic operations such as hashings, encryptions, key exchanges or signatures. For example, in Zerocash [BCG+14], the authors considered proving the SHA256 hash computation to build a privacy-preserving cryptocurrency protocol. Zcash (ZEC) is a cryptocurrency that first implemented the Zerocash protocol with many improvements over the years. Among these, they replaced the SHA256 hash by a variant of the Pedersen hash based on elliptic curves. Gyges [JKS16] and Hawk [KMS+16] considered proving the RSA encryption for privacy-preserving blockchain smart contracts. Cinderella [DFKP16] considered proving the validity and compliance of X.509 certificates by proving the RSA signature verification. Later, CØCØ's authors observed that the RSA scheme is not a well suited computation to be proven in a SNARK and considered replacing it by a hybrid encryption. They first proved an ECDH key exchange and then a lightweight symmetric encryption (Speck [BSS+13] and Chaskey-LTS [MMV+14]). More recently, zk-rollups were proposed to solve the scalability problem of the Ethereum blockchain. A zk-rollup operator validates a batch of transactions by verifying a proof of the correct verifications of many signatures (e.g. ConsenSys rollup and Polygon Hermez rollup verify EdDSA signatures on a twisted Edwards curve associated to BN254).

Which elliptic curve to choose? Some of these computations are elliptic curve based cryptographic primitives. They require an elliptic curve E_0 to express the computation, independently of the SNARK elliptic curve E to prove the computation. CØCØ's authors were the first to propose to construct a customized elliptic curve to efficiently express this kind of computations.

Proof systems use different models to translate a generic-purpose computation into an equivalent arithmetized statement. The R1CS (cf. Sec. 6.1) is one of the most used arithmetization models. A key property of this model is that multiplications, squarings and inversions have the same cost while additions and subtractions are free. Therefore, an implementor should express the statement they want to prove with the lowest multiplicative complexity and, counter-intuitively, they should bear in mind that inversions are not costly. The arithmetization step takes place in $\mathbb{F}_r$ where r is the prime subgroup order of the SNARK curve, i.e. $r \mid \#E$. CØCØ constructs a custom curve E_0 precisely defined over that $\mathbb{F}_r$ to avoid emulating non-native field operations (cf. Fig. 2.4). Following the guidelines described when constructing `curve25519`, the authors proposed the curve $E_0(\mathbb{F}_r)$ in Montgomery form $y^2 = x^3 + 126932x^2 + x$. The group $E_0(\mathbb{F}_r)$ has order 8×251-bit prime and its quadratic twist has order 4×252-bit. Implementing affine-coordinates

scalar multiplication using this curve is very efficient in the R1CS model. Later, Zcash engineers proposed the use of a twisted Edwards curve that is more efficient than the CØCØ Montgomery curve. It was built on top of the BLS12-381 curve and named Jubjub.

Scalar multiplication in R1CS. In fact, conditional statements are costly in R1CS and one resorts to a *double-and-always-add*-like algorithm (as in a side-channel resistant implementation) instead of the classical *double-and-add* algorithm (Alg. 1.1) to express a scalar multiplication. At first sight, it would seem that windowed multiplications would be expensive in this context but it turns out that constant-windowed methods can be achieved efficiently in R1CS using a twisted Edwards curve. The affine group law on these curves is *complete* meaning that one can use a lookup table to select the precomputed points (x_i, y_i) or the zero point $(x_0, y_0) = (0, 1)$ to be added in the algorithm. This is done through polynomials which vanish at the inputs that are not being selected. The optimal lookup is a 2-bit input 4-value output loopkup. We show how to perform a 2-bit windowed scalar multiplication in R1CS in Algorithms 6.2 and 6.1. With this technique our implementation in `gnark` costs 3060 constraints for a variable-base scalar multiplication and 2436 constraints for a fixed-base multiplication.

Algorithm 6.1: `Lookup2`: 2-bit lookup table in R1CS
Input: bits (b_0, b_1), and constants (c_0, c_1, c_2, c_3)
Output: $r = \begin{cases} c_0, & \text{if } b_0 = 0, b_1 = 0 \\ c_1, & \text{if } b_0 = 0, b_1 = 1 \\ c_2, & \text{if } b_0 = 1, b_1 = 0 \\ c_3, & \text{if } b_0 = 1, b_1 = 1 \end{cases}$

$t_1, t_2 \leftarrow$ temporary variables;
$(c_3 - c_2 - c_1 + c_0) \times b_1 = t_1 - c_1 + c_0$; // Gate 1
$t_1 \times b_0 = t_2$; // Gate 2
$(c_2 - c_0) \times b_1 = r - t_2 - c_0$; // Gate 3
return r;

Algorithm 6.2: 2-bit windowed scalar multiplication in R1CS
Input: a scalar s and a point $P \in E_0$
Output: $R = [s]P \in E_0$

$(b_0, \cdots, b_{n-1}) \leftarrow$ the binary decomposition of s (b_0 being the least significant bit and assuming n is even) ; // This is also a statement to prove
$R, T \in E_0$; $A \leftarrow [2]P$; $B \leftarrow A + P$;
$R_x \leftarrow$ `Lookup2`$(b_{n-1}, b_{n-2}, 0, P_x, A_x, B_x)$;
$R_y \leftarrow$ `Lookup2`$(b_{n-1}, b_{n-2}, 1, P_y, A_y, B_y)$;
for b_i *from* b_{n-3} *downto* b_0, $i \leftarrow i - 2$**:**
  $R \leftarrow [4]R$;
  $T_x \leftarrow$ `Lookup2`$(b_i, b_{i-1}, 0, P_x, A_x, B_x)$;
  $T_y \leftarrow$ `Lookup2`$(b_i, b_{i-1}, 1, P_y, A_y, B_y)$;
  $R \leftarrow R + T$;
return R;

GLV in R1CS. In [MSZ21] Masson et al. performed an exhaustive search of a twisted Edwards curve defined over the scalar field of BLS12-381 with a small Complex Multiplication discriminant in order to speed up the scalar multiplication using a fast endomorphism. They found an isolated curve with the discriminant $-D = -8$, called Bandersnatch. It has an efficient degree 2 endomorphism $\phi \in \mathcal{O}_{-8}$. Given in twisted Edwards form and projective coordinates, its equation is

$$\phi(x, y, z) = (f(y)h(y), g(y)xy, h(y)xy)$$

where $f(y) = c(z^2 - y^2)$, $g(y) = b(y^2 + bz^2)$ and $h(y) = y^2 - bz^2$ with

$b =$ `0x52c9f28b828426a561f00d3a63511a882ea712770d9af4d6ee0f014d172510b4`,
$c =$ `0x6cc624cf865457c3a97c6efd6c17d1078456abcfff36f4e9515c806cdf650b3d`.

We noted that this curve has a non-square coefficient $a = -5$, hence the formulas are *incomplete*. However, all points at infinity are of even order hence checking that the points are of the right odd prime order is enough to rule out exceptions [HWCD08, Th. 1]. Our implementation in `gnark` costs 2436 constraints for a variable-base scalar multiplication and 1914 constraints for a fixed-base multiplication while, in [MSZ21], the authors report 2608 constraints for the variable-base case using the `arkworks` ecosystem.

The key difference is in the implementation of the scalar decomposition in the GLV method. The sub-scalars are R1CS variables that lie in $\mathbb{F}_r$, so if one of these is negative it will be reduced modulo r (the SNARK curve subgroup order) and not m (the Bandersnatch curve subgroup order), resulting in an incorrect computation. We observed that the short basis determinant in this case is negative and by always Babai-rounding above the discriminant, we force the first sub-scalar to be negative. On the one hand, we compute the scalar decomposition outside of the R1CS circuit and only verify it inside — costing less constraints. On the other hand, we now know in advance which scalar is negative so we negate the corresponding point instead. However this trick increases the sub-scalars size bounds by one bit. This might be of independent interest for side-channel resistant GLV scalar multiplication out-circuit.

All SNARK curves can have an associated twisted Edwards curve but it is unlikely to find such a curve with a small CM discriminant. In Table 6.1 we gather, for the main SNARK curves discussed in this thesis, their associated twisted Edwards curve(s). Note that the multiplication by the curve coefficients is free in the R1CS model as they are constants with respect to the statement.

SNARK curve	associated twisted Edwards curve $ax^2 + y^2 = 1 + dx^2y^2$			
	$a = -1$?	curve order	twist cofactor	fast endomorphism?
BN254 (Ethereum)	✓	$8 \times$ (251-bit prime)	4	✗
BLS12-381	✓ (Jubjub)	$8 \times$ (252-bit prime)	4	✗
	✗($a = -5$) (Bandersnatch)	$4 \times$ (253-bit prime)	$2^7 \cdot 3^3$	✓
BLS24-317	✓	$8 \times$ (250-bit prime)	1	✗
BLS12-377	✓	$4 \times$ (251-bit prime)	8	✗
BW6-761	✓	$8 \times$ (374-bit prime)	4	✗
BLS24-315	✓	$8 \times$ (250-bit prime)	4	✗
BW6-633	✓	$8 \times$ (312-bit prime)	4	✗
MNT4-753	✓	$8 \times$ (750-bit prime)	4	✗
MNT4-298	✓	$4 \times$ (296-bit prime)	8	✗

Table 6.1: Main SNARK curves and their associated twisted Edwards curves.

Chapter

7

Pairings in rank-1 constraint systems

Bilinear pairings have been used in different cryptographic applications and demonstrated to be a key building block for a plethora of constructions such as SNARKs. In particular, we have demonstrated in Chapter 4 that pairing-based SNARKs are suitable for proof recursion through 2-chains of elliptic curves. In this scenario one requires to express efficiently a pairing computation as a SNARK arithmetic circuit. Other compelling applications such as verifying Boneh–Lynn–Shacham (BLS) signatures [BLS01] or KZG polynomial commitment opening in a SNARK fall into the same requirement. The implementation of pairings is challenging but the literature has very detailed approaches on how to reach practical and optimized implementations in different contexts and for different target environments. However, to the best of our knowledge, no previous publication has addressed the question of efficiently implementing a pairing as a SNARK arithmetic circuit.

In this chapter, we consider efficiently implementing pairings in R1CS. We show that our techniques almost halve the arithmetic circuit depth of the previously best known pairing implementation on a BLS12 curve, resulting in 70% faster proving time. We also investigate and implement the case of BLS24 curves. This chapter, in part, is a reprint of the material as it appears in our submitted work [Hou22].

7.1 Pairings out-circuit

We recall the best algorithms from the literature to implement efficiently a pairing over two families of elliptic curves BLS12 and BLS24.

7.1.1 Pairings over inner BLS12 and BLS24 curves.

Table 7.1 summarizes the salient parameters of BLS12 and BLS24 curves and Table 7.2 gives the concrete parameters of the curves we suggested in [EHG22] and their security, namely the BLS12-377 and BLS24-315. Next we will focus on efficient Miller loop computation and final exponentiation from the literature for these curves. The most efficient

Table 7.1: Polynomial parameters of BLS12 and BLS24 families.

Family	k	$-D$	ρ	$r(x)$	$p(x)$	$t(x)$
BLS12	12	-3	3/2	x^4-x^2+1	$(x^6-2x^5+2x^3+x+1)/3+x$	$x+1$
BLS24	24	-3	5/4	x^8-x^4+1	$(x^{10}-2x^9+x^8-x^6+2x^5-$ $x^4+x^2+x+1)/3$	$x+1$

Table 7.2: Security level estimates of BLS12-377 and BLS24-315 curves from [BCG+20, EHG22], with seeds $x_{377}=$ `0x8508c00000000001`, $x_{315}=-$`0xbfcfffff`,

curve	k	$-D$	ref	r bits	p bits	p^k bits	DL cost in $\mathbb{F}_{p^k}$
BLS12-377, x_{377}	12	-3	[BCG+20]	253	377	4521	2^{126}
BLS24-315, x_{315}	24	-3	[EHG22, Tab. 10]	253	315	7543	2^{160}

pairing on BLS curves is the optimal ate pairing [Ver10] (cf. Sec. 1.2). Given $P\in\mathbb{G}_1$ and $Q\in\mathbb{G}_2$, it consists in computing

$$e(P,Q)=f_{x,Q}(P)^{(p^k-1)/r}$$

where x is the curve's seed and k the curve's embedding degree (12 for BLS12 and 24 for BLS24). The Miller loop computation (Alg. 1.2) boils down to $\mathbb{G}_2$ arithmetic ($[2]S$ and $S+Q$), line computations and evaluations in $\mathbb{F}_{p^k}$ ($\ell_{S,S}(P)$ and $\ell_{S,Q}(P)$), squarings in $\mathbb{F}_{p^k}$ (m^2) and sparse multiplications in $\mathbb{F}_{p^k}$ ($m\cdot\ell$). The vertical lines ($v_{[2]S}(P)$ and $v_{S+Q}(P)$) are ignored because eliminated later by the final exponentiation ($k=12$ or 24 is even). These operations are best optimized following [ABLR14] for a single pairing and [Sco19] for a multi-pairing.

$\mathbb{F}_{p^k}$ towering and arithmetic. The extension field $\mathbb{F}_{p^k}$ can be constructed in different ways. A pairing-friendly towering is built using a sequence of quadratic and cubic extension fields. An appropriate choice of irreducible polynomials is recommended to efficiently implement Karatsuba [KO63] and Chung–Hasan formulas [CH07]. The tower $\mathbb{F}_{p^{12}}$ can be built as

$$\mathbb{F}_p \xrightarrow{u^2-\alpha} \mathbb{F}_{p^2} \xrightarrow{v^3-\beta} \mathbb{F}_{p^6} \xrightarrow{w^2-\gamma} \mathbb{F}_{p^{12}} \quad \text{or} \quad \mathbb{F}_p \xrightarrow{u^2-\alpha} \mathbb{F}_{p^2} \xrightarrow{v^2-\beta} \mathbb{F}_{p^4} \xrightarrow{w^3-\gamma} \mathbb{F}_{p^{12}}.$$

Both options have $\mathbb{F}_{p^2}$ as a sub-field, needed to compress $\mathbb{G}_2$ coordinates. The arithmetic on the first option is usually slightly faster while the second one allows a better compression ratio (1/3 instead of 1/2) for $\mathbb{G}_T$ elements via XTR or CEILIDH [Sta21] (instead of Lucas or $\mathbb{T}_2$ [Sta21]). The tower $\mathbb{F}_{p^{24}}$ can be built as

$$\mathbb{F}_p \xrightarrow{u^2-\alpha} \mathbb{F}_{p^2} \xrightarrow{v^2-\beta} \mathbb{F}_{p^4} \xrightarrow{w^3-\gamma} \mathbb{F}_{p^{12}} \xrightarrow{i^2-\delta} \mathbb{F}_{p^{24}} \quad \text{or} \quad \mathbb{F}_p \xrightarrow{u^2-\alpha} \mathbb{F}_{p^2} \xrightarrow{v^2-\beta} \mathbb{F}_{p^4} \xrightarrow{w^2-\gamma} \mathbb{F}_{p^8} \xrightarrow{i^3-\delta} \mathbb{F}_{p^{24}}.$$

The same remarks apply to the towering options here, this time with $\mathbb{F}_{p^4}$ as the sub-field needed to compress $\mathbb{G}_2$ coordinates for BLS24.

$\mathbb{G}_2$ arithmetic and line evaluations. It was shown in [CLN10, AKL+11, GAL+13, ABLR14] that the choice of homogeneous projective coordinates is advantageous at the 128-bit security level. This is due to the large inversion/multiplication ratio and the possibility to maximize the shared intermediate computations between the $\mathbb{G}_2$ arithmetic and the line evaluations. In [ABLR14], the authors also suggest to multiply the line by w^3 (in case of $\mathbb{F}_{p^{12}}$ towering for instance) which is eliminated later by the final exponentiation. This is to obtain a fast sparse multiplication by the lines. Given $S = (X_S, Y_S, Z_S) \in \mathbb{G}_2 \cong E'[r](\mathbb{F}_{p^{k/d}})$, the derived formulas are

$$X_{[2]S} = X_S Y_S (Y_S^2 - 9b'Z_S^2)/2;\ Y_{[2]S} = ((Y_S^2 + 9b'Z_S^2)/2)^2 - 27b'Z_S^4;\ Z_{[2]S} = 2Y_S^3 Z_S\ .$$

When the curve has a D-type twist given by the twisting isomorphism $\psi : E'(\mathbb{F}_{p^{k/d}}) \to E(\mathbb{F}_{p^k})$, the tangent line evaluated at (x_P, y_P) can be computed with

$$g_{[2]\psi(S)}(P) = -2Y_S Z_S \cdot y_P + 3X_S^2 \cdot x_P w + (3b'Z_S^2 - Y_S^2)w^3\ .$$

Similarly, if $S = (X_S, Y_S, Z_S)$ and $Q = (x_Q, y_Q) \in E'(\mathbb{F}_{p^{k/d}})$ are points in homogeneous projective and affine coordinates, respectively, one can compute the mixed addition $S + Q$ as follows

$$X_{S+Q} = \lambda(\lambda^3 + Z_S\theta^2 - 2X_S\lambda^2);\ Y_{S+Q} = \theta(3X_S\lambda^2 - \lambda^3 - Z_S\theta^2) - Y_S\lambda^3;\ Z_{S+Q} = Z_S\lambda^3$$

where $\theta = Y_S - y_Q Z_S$ and $\lambda = X_S - x_Q Z_S$. In the case of a D-type twist for example, the line evaluated at $P = (x_P, y_P)$ can be computed with

$$g_{\psi(S+Q)}(P) = -\lambda y_P - \theta x_P w + (\theta x_2 - \lambda y_2)w^3\ .$$

For multi-pairings $\prod_{i=0}^{n-1} e(P_i, Q_i)$, one can share the squaring $m^2 \in \mathbb{F}_{p^k}$ between the different pairs. Scott [Sco19] further suggested storing and then multiplying together the lines ℓ 2-by-2 before multiplying them by the Miller loop accumulator m. This fully exploits any sparsity which may exist in either multiplicand.

The final exponentiation. After the Miller loop, an exponentiation in $\mathbb{F}_{p^k}$ to the fixed $(p^k - 1)/r$ is necessary to ensure the output uniqueness of the (optimal) ate (and Tate) pairings. For BLS curves, many works have tried to speed this computation up by applying vectorial addition chains or lattice-based reduction approaches [HHT20, AFK+13, GF16]. It is usually divided into an easy part and a hard part, as follows:

$$\begin{aligned} \frac{p^k - 1}{r} &= \underbrace{\frac{p^k - 1}{\Phi_k(p)}}_{\text{easy part}} \cdot \underbrace{\frac{\Phi_k(p)}{r}}_{\text{hard part}} \\ &= \underbrace{(p^d - 1)\frac{\sum_{i=0}^{e-1} p^{id}}{\Phi_k(p)}}_{\text{easy part}} \cdot \underbrace{\frac{\Phi_k(p)}{r}}_{\text{hard part}} \end{aligned} \tag{7.1}$$

where Φ_k is the k-th cyclotomic polynomial and $k = d \cdot e$. For BLS12 and BLS24 curves ($d = 6$, and $k = 12$ resp. $k = 24$), the easy part happens to be $(p^{k/2} - 1)(p^{k/d} + 1)$. It is made of Frobenius powers, two multiplications and a single inversion in $\mathbb{F}_{p^k}$. The most

efficient algorithms for the hard part stem from [HHT20], which we suggest to implement as in Alg. 7.1 and Alg. 7.2 ($3 \cdot \Phi_k(p)/r$).

Algorithm 7.1: Final exp. hard part for BLS12 curves.
Input: $m = f_{x,Q}(P) \in \mathbb{F}_{p^{12}}$
Output: $m^{3\cdot\Phi_{12}(p)/r} \in \mathbb{G}_T$

$t_0 \leftarrow m^2$
$t_1 \leftarrow m^x$ // exp. by the fixed seed x
$t_2 \leftarrow \bar{m}$ // conjugate
$t_1 \leftarrow t_1 \cdot t_2$
$t_2 \leftarrow t_1^x$
$t_1 \leftarrow \bar{t_1}$
$t_1 \leftarrow t_1 \cdot t_2$
$t_2 \leftarrow t_1^x$
$t_1 \leftarrow t_1^p$ // Frob.
$t_1 \leftarrow t_1 \cdot t_2$
$m \leftarrow m \cdot t_0$
$t_0 \leftarrow t_1^x$
$t_2 \leftarrow t_0^x$
$t_0 \leftarrow t_1^{p^2}$ // Frob. square
$t_1 \leftarrow \bar{t_1}$
$t_1 \leftarrow t_1 \cdot t_2$
$t_1 \leftarrow t_1 \cdot t_0$
$m \leftarrow m \cdot t_1$
return m

Algorithm 7.2: Final exp. hard part for BLS24 curves.
Input: $m = f_{x,Q}(P) \in \mathbb{F}_{p^{24}}$
Output: $m^{3\cdot\Phi_{24}(p)/r} \in \mathbb{G}_T$

$t_0 \leftarrow m^2$
$t_1 \leftarrow m^x$ // exp. by the fixed seed x
$t_2 \leftarrow \bar{m}$ // conjugate
$t_1 \leftarrow t_1 \cdot t_2$
$t_2 \leftarrow t_1^x$
$t_1 \leftarrow \bar{t_1}$
$t_1 \leftarrow t_1 \cdot t_2$
$t_2 \leftarrow t_1^x$
$t_1 \leftarrow t_1^p$ // Frob.
$t_1 \leftarrow t_1 \cdot t_2$
$m \leftarrow m \cdot t_0$
$t_0 \leftarrow t_1^x$
$t_2 \leftarrow t_0^x$
$t_0 \leftarrow t_1^{p^2}$ // Frob. square
$t_2 \leftarrow t_0 \cdot t_2$
$t_1 \leftarrow t_2^x$
$t_1 \leftarrow t_1^x$
$t_1 \leftarrow t_1^x$
$t_1 \leftarrow t_1^x$
$t_0 \leftarrow t_2^{p^4}$ // Frob. quad
$t_0 \leftarrow t_0 \cdot t_1$
$t_2 \leftarrow \bar{t_2}$
$t_0 \leftarrow t_0 \cdot t_2$
$m \leftarrow m \cdot t_0$
return m;

Since the elements are in a cyclotomic subgroup after the easy part exponentiation, the squarings are usually implemented using the Granger–Scott method [GS10]. The dominating cost of the hard part is the exponentiation to the fixed seed $m \mapsto m^x$ which is usually implemented with a short addition chain of plain multiplications and cyclotomic squarings. Further savings, when the seed is even [GF16], do not apply to inner BLS because the seed is always odd ($x \equiv 1 \mod 3 \cdot 2^L$).

7.1.2 Torus-based cryptography

An algebraic torus is a type of commutative affine algebraic group that we will need in optimizing the pairing computation in R1CS. Here we give a basic definition and some useful results from [RS03] and [NBS08].

Definition 7.1. *The norm of an element $\alpha \in \mathbb{F}_{p^k}$ with respect to $\mathbb{F}_p$ is defined as* $N_{\mathbb{F}_{p^k}/\mathbb{F}_p} =$

$\alpha\alpha^p \cdots \alpha^{p^{k-1}} = \alpha^{(p^k-1)/(p-1)}$. *For a positive integer* k *and a subfield* $F \subset \mathbb{F}_{p^k}$*, the torus is*

$$T_k(\mathbb{F}_p) = \bigcap_{\mathbb{F}_p \subseteq F \subset \mathbb{F}_{p^k}} \ker(N_{\mathbb{F}_{p^k}/F})$$

In this case, $F = \mathbb{F}_{p^d}$ *for* $d \mid k$ *and* $N_{\mathbb{F}_{p^k}/\mathbb{F}_{p^d}} = \alpha^{(p^k-1)/(p^d-1)}$*. Thus, equivalently, we have*

$$T_k(\mathbb{F}_p) = \{\alpha \in \mathbb{F}_{p^k} | \alpha^{(p^k-1)/(p^d-1)} = 1\} \text{ and } |T_k(\mathbb{F}_p)| = \Phi_k(p) \ .$$

Lemma 7.1 ([NBS08, Lemma 1])**.** *Let* $\alpha \in \mathbb{F}^*_{p^k}$*, then* $\alpha^{(p^k-1)/\Phi_k(p)} \in T_k(\mathbb{F}_p)$.

Lemma 7.2 ([NBS08, Lemma 2])**.** $d \mid k \implies T_k(\mathbb{F}_p) \subseteq T_{k/d}(\mathbb{F}_{p^d})$.

Corollary 7.1. *After the easy part of the final exponentiation in the pairing computation (Eq. 7.1), elements are in the torus* $\mathbb{T}_k(\mathbb{F}_p)$ *and thus in each torus* $\mathbb{T}_{k/d}(\mathbb{F}_{p^d})$ *for* $d \mid k$*,* $d \neq k$*.*

7.1.3 $\mathbb{T}_2$ cryptosystem

After the easy part of the final exponentiation the elements are in a proper subgroup of $\mathbb{F}_{p^k}$ that coincides with some algebraic tori as per Corollary 7.1. Rubin and Silverberg introduced a torus-based cryptosystem in [RS03], called $\mathbb{T}_2$.

Let $q = p^{k/2}$ (q odd) and $\mathbb{F}_{q^2} = \mathbb{F}_q[w]/(w^2 - \gamma)$. Let $G_{q,2} = \{m \in \mathbb{F}_{q^2} | m^{q+1} = 1\}$, which means that if $m = m_0 + wm_1 \in G_{q,2}$ then $m_0^2 - \gamma m_1^2 = 1$. This norm equation characterizes the cyclotomic subgroup where the result of the easy part lies. When $m_1 = 0$, then m_0 must be 1 or -1. The authors define the following compression/decompression maps on $G_{q,2} \setminus \{-1, 1\}$

$$\begin{aligned} \texttt{Compress} \quad \zeta : G_{q,2} \setminus \{-1,1\} &\to \mathbb{F}^*_q \\ m &\mapsto \frac{1+m_0}{m_1} = g; \\ \texttt{Decompress} \quad \zeta^{-1} : \mathbb{F}^*_q &\to G_{q,2} \setminus \{-1,1\} \\ g &\mapsto \frac{g+w}{g-w} \ . \end{aligned}$$

In $\mathbb{T}_2$-cryptography, one compresses $G_{q,2} \setminus \{-1, 1\}$ elements into $\mathbb{F}^*_q$ (half their size) using ζ and performs all the arithmetic in $\mathbb{F}^*_q$ without needing to decompress back into $G_{q,2}$ (ζ^{-1}). Given $g, g' \in \mathbb{F}^*_q$ where $g \neq -g'$, one defines the multiplication as

$$\texttt{Multiply} \quad (g, g') \mapsto \frac{g \cdot g' + \gamma}{g + g'} \ .$$

One can derive other operations in compressed form such as

$$\begin{aligned} \texttt{Inverse} \quad & g \mapsto -g; \\ \texttt{Square} \quad & g \mapsto \tfrac{1}{2}(g + \gamma/g); \\ \texttt{Frobenius map} \quad & g \mapsto \frac{g^{p^i}}{\gamma^{(p^i-1)/2}} \ . \end{aligned}$$

7.2 Pairings in-circuit

In subsection 7.1, we presented results from the literature that yield the most efficient pairing computation on BLS12 and BLS24 curves. Porting these results mutatis-mutandis to the R1CS model would result in a circuit of approximately 80000 multiplication gates in the case of the BLS12-377 curve. Here, we present an algorithm and implementation that yield the smallest number of constraints so far in the literature (around 11500 for the BLS12-377 curve). In the sequel, we will denote by $\mathtt{C}$ the number of multiplication gates and take mainly the example of a BLS12 curve.

7.2.1 Miller loop

$\mathbb{G}_2$ ***arithmetic.*** Since inversions cost almost as much as multiplications in R1CS, it is better to use affine coordinates in the Miller loop. Over $\mathbb{F}_p$ (base field of the inner BLS12 which is the SNARK field of the outer BW6 curve), an inversion $1/x = y$ costs $2\mathtt{C}$. First $1\mathtt{C}$ for the multiplication $x \cdot y$ where y is provided as an input and then $1\mathtt{C}$ for the equality check $x \cdot y \stackrel{?}{=} 1$. For division, instead of computing an inversion and then a multiplication as it is custom, one would compute directly the division in R1CS. The former costs $3\mathtt{C}$ while the later costs $2\mathtt{C}$ as for $x/z = y \implies x \stackrel{?}{=} z \cdot y$. A squaring costs as much as a multiplication over $\mathbb{F}_p$ ($x = y$).

The same observations work over extension fields $\mathbb{F}_{p^e}$ except for squaring where the Karatsuba technique can be specialized. For example over $\mathbb{F}_{p^2}$, a multiplication costs $3\mathtt{C}$, a squaring $2\mathtt{C}$, an inversion and a division $5\mathtt{C}$ ($2\mathtt{C}$ for the equality check).

Point doubling and addition in affine coordinates is as follows:

Double: $[2](x_S, y_S) = (x_{[2]S}, y_{[2]S})$	*Add*: $(x_S, y_S) + (x_Q, y_Q) = (x_{S+Q}, y_{S+Q})$
$\lambda = 3x_S^2/2y_S$	$\lambda = (y_S - y_Q)/(x_S - x_Q)$
$x_{[2]S} = \lambda^2 - 2x_S$	$x_{S+Q} = \lambda^2 - x_S - x_Q$
$y_{[2]S} = \lambda(x_S - x_{[2]S}) - y_S$	$y_{S+Q} = \lambda(x_Q - x_{S+Q}) - y_Q$

For BLS12 curves, $\mathbb{G}_2$ coordinates are over $\mathbb{F}_{p^2}$ and Table 7.3 summarizes the cost of $\mathbb{G}_2$ arithmetic in R1CS. Note that a doubling is more costly in R1CS than an addition

Table 7.3: $\mathbb{G}_2$ arithmetic cost in R1CS over $\mathbb{F}_{p^2}$

	Div ($5\mathtt{C}$)	Square ($2\mathtt{C}$)	Mul ($3\mathtt{C}$)	total
Double	1	2	1	$12\mathtt{C}$
Add	1	1	1	$10\mathtt{C}$

because the tangent slope λ requires a squaring and a division instead of just a division. The Miller functions parameter is constant — the seed-x for BLS. Counter-intuitively in this case, we generate a short addition chain that maximizes the number of additions instead of doublings using the `addchain` Software from McLoughlin: `https://github.com/mmcloughlin/addchain`.

It turns out we can do better: when the seed x bit is 1, a doubling and an addition $[2]S + Q$ ($22\mathtt{C}$) is computed but instead we can compute $(S + Q) + S$ which costs $20\mathtt{C}$. Moreover, we can omit the computation of the y-coordinate of $S + Q$ as pointed out in a

different context in [ELM03].

Double-and-Add: $[2](x_S, y_S) + (x_Q, y_Q) = (x_{(S+Q)+S}, y_{(S+Q)+S})$

$$\begin{aligned}
\lambda_1 &= (y_S - y_Q)/(x_S - x_Q) \\
x_{S+Q} &= \lambda_1^2 - x_S - x_Q \\
\lambda_2 &= -\lambda_1 - 2y_S/(x_{S+Q} - x_S) \\
x_{(S+Q)+S} &= \lambda_2^2 - x_S - x_{S+Q} \\
y_{(S+Q)+S} &= \lambda_2(x_S - x_{(S+Q)+S}) - y_S
\end{aligned}$$

which costs 17C in total (2 Div, 2 Square and 1 Mul).

Line evaluations. For BLS12, a line ℓ in $\mathbb{F}_{p^2}$ is of the form $ay + bx + c = 0$. In the Miller loop, we need to compute the lines that go through the untwisted $\mathbb{G}_2$ points $[2]S$ and $S + Q$ and to evaluate them at $P \in \mathbb{G}_1$. That is, $\ell_{\psi([2]S)}(P)$ and $\ell_{\psi(S+Q)}(P)$ where $\psi : E'(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^{12}})$ is the untwisting isomorphism. Following [ABLR14], both lines are sparse elements in $\mathbb{F}_{p^{12}}$ of the form $ay_P + bx_P \cdot w + c \cdot w^3$ with $a, b, c \in \mathbb{F}_{p^2}$. In R1CS, we precompute $1/y_P$ and x_P/y_P for 1C each and represent the lines by $1+b'x_P/y_P \cdot w + c'/y_P \cdot w^3$. This does not change the final result because $1/a$ is in a proper subfield of $\mathbb{F}_{p^k}$. A full multiplication in $\mathbb{F}_{p^{12}}$ costs 54C and a sparse multiplication as in [ABLR14] costs 39C, while with this representation it costs only 30C with a single 2C precomputation.

We adapt the "$\mathbb{G}_2$ arithmetic and line evaluations" formulas from the previous section (pairing out-circuit) to the affine setting together with the optimizations in this section.

Let $S = (x_S, y_S)$, $Q = (x_Q, y_Q) \in \mathbb{G}_2 \cong E'[r](\mathbb{F}_{p^{k/d}})$ and $P = (x_P, y_P) \in E[r](\mathbb{F}_p)$. For a D-type twist, in the double step, the tangent line to S evaluated at P is computed with

$$g_{[2]\psi(S)}(P) = 1 - \lambda \cdot x_P/y_P w + (\lambda x_S - y_S)/y_P \cdot w^3$$

where $\lambda = 3x_P^2/2y_P$.

In the double-and-add step, the line through S and Q evaluated at P is computed with

$$g_{\psi(S+Q)}(P) = 1 - \lambda_1 \cdot x_P/y_P w + (\lambda_1 x_S - y_S)/y_P \cdot w^3$$

and the line through $S + Q$ and S is computed with

$$g_{\psi((S+Q)+S)}(P) = 1 - \lambda_2 \cdot x_P/y_P w + (\lambda_2 x_S - y_S)/y_P \cdot w^3$$

where $\lambda_1 = (y_Q - y_S)/(x_Q - x_S)$ and $\lambda_2 = -\lambda_1 - 2y_S/(x_{S+Q} - x_S)$.

$\mathbb{F}_{p^k}$ towering and arithmetic. For the towering of $\mathbb{F}_{p^{12}}$, we choose the option where $\mathbb{F}_{p^{12}}$ is a quadratic extension of $\mathbb{F}_{p^6}$ to be able to use $\mathbb{T}_2$ arithmetic as we will show later. The arithmetic costs in terms of constraints are summarized in Table 7.4.

Table 7.4: $\mathbb{F}_{p^{12}}$ arithmetic cost in R1CS

	Mul	Square	Div	sparse Mul
$\mathbb{F}_{p^{12}}$	54C	36C	66C	30C

7.2.2 Final exponentiation

Easy part. The easy part (Eq. 7.1) consists in raising the Miller loop output $m \in \mathbb{F}_{p^{12}}$ to the power $(p^6-1)(p^2+1)$, which is usually implemented as follows

$$\begin{array}{rcll} t & \leftarrow & \bar{m} & (0\mathtt{C}) \\ m & \leftarrow & 1/m & (66\mathtt{C}) \\ t & \leftarrow & t \cdot m & (54\mathtt{C}) \\ m & \leftarrow & t^{p^2} & (0\mathtt{C}) \\ m & \leftarrow & t \cdot m & (54\mathtt{C}) \end{array}$$

where $t \in \mathbb{F}_{p^{12}}$ is a temporary variable. The conjugate $\bar{m}$ and the Frobenius map t^{p^2} are essentially free because they only involve multiplications by constants. We further merge the inversion (66C) and the multiplication (54C) in a division operation (66C). The total cost is 120C instead of 174C.

Hard part. The most efficient implementation is described in Alg. 7.1. Only the multiplications and cyclotomic squarings increase the number of constraints. Squarings in cyclotomic subgroups are well studied in the literature and in Table 7.5 we give the best algorithms in the R1CS model. It can be seen that for a single square or two squares in a row, Granger-Scott algorithm [GS10] is preferred while compression-based methods are better for other cases. For 3 squares in a row the SQR12345 variant of the Karabina method [Kar13] is preferred while for more than 4 the SQR234 variant yields the smallest number of constraints. Usually, out-circuit, we would use the Granger-Scott method because of the inversion cost in the decompression due to Karabina method but in R1CS inversions are not costly.

Table 7.5: Squaring costs in the cyclotomic subgroup of $\mathbb{F}_{p^{12}}$ in R1CS

	Compress	Square	Decompress
Karatsuba + Chung–Hasan	0	36C	0
Granger-Scott [GS10]	0	18C	0
Karabina [Kar13] (SQR2345)	0	12C	19C
Karabina [Kar13] (SQR12345)	0	15C	8C

$\mathbb{T}_2$ ***arithmetic.*** Corollary 7.1 states that after the easy part of the final exponentiation, the result lies in $\mathbb{T}_2(\mathbb{F}_{p^6})$ and thus $\mathbb{T}_2$ arithmetic can be used to further reduce the number of constraints in the hard part. We first compress the element, use squarings and multiplications in the compressed form and finally decompress the result following the cost in Table 7.6. The $\mathbb{T}_2$ formulas are well defined over $G_{q,2} \setminus \{-1, 1\}$ but for pairings we only consider $G_{q,2} \setminus \{1\}$ as both exception values are mapped to 1 after the final exponentiation. We can even get rid of the one-time cost of compression and decompression. First, the decompression is not needed as the applications we are interested in do not require the

Table 7.6: $\mathbb{T}_2$ arithmetic cost in R1CS.

	Compress	Square	Mul	Decompress
$\mathbb{T}_2$	24C	24C	42C	48C

exact value of the pairing but just to check a multi-pairing equation, *i.e.* $\prod_{i=0}^{n-1} e(P_i, Q_i) \stackrel{?}{=} 1$. In this case, the equality check can be performed in the compressed form costing even less constraints (kC vs. $k/2$C). For the compression, it can be absorbed in the easy part computation as it was shown in [NBS08]. Let $m = m_0 + wm_1 \in \mathbb{F}_{p^{12}}$ be the Miller loop result. We do not consider the exception case $m = 1$ as this would mean that the points are co-linear which is not the case for pairs correctly in $\mathbb{G}_1$ and $\mathbb{G}_2$ (we assume this is verified out-circuit). The easy part is $m^{(p^6-1)(p^2+1)}$ where

$$\begin{aligned} m^{p^6-1} &= (m_0 + wm_1)^{p^6-1} \\ &= (m_0 + wm_1)^{p^6}/(m_0 + wm_1) \\ &= (m_0 - wm_1)/(m_0 + wm_1) \\ &= (-m_0/m_1 + w)/(-m_0/m_1 - w) \end{aligned}$$

which means $\zeta(m^{(p^6-1)}) = -m_0/m_1$. Hence we can absorb the $\mathbb{T}_2$ compression cost when carrying this the exponentiation to $p^6 - 1$ and continue with the exponentiation to $p^2 + 1$ in the compressed form

$$\begin{aligned} \zeta(m^{(p^6-1)})^{p^2+1} &= (-m_0/m_1)^{p^2+1} \\ &= \underbrace{\underbrace{(-m_0/m_1)^{p^2}}_{\mathbb{T}_2\texttt{-Frobenius map}} \cdot(-m_0/m_1)}_{\mathbb{T}_2\texttt{-Multiply}} \end{aligned}$$

This costs only 60C in comparison of 120C previously. In [NBS08], the authors noted that one can perform the whole Miller loop in $\mathbb{T}_2$. The original motivation was to compress the computation for constrained execution environments but in our case the motivation would be to benefit from the $\mathbb{T}_2$ arithmetic that costs less R1CS constraints than the plain computation. However, having to deal with the exception case $m = 1$ separately is very costly in R1CS. In fact, conditional statements are carried through polynomials which vanish at the inputs that are not being selected. As an example, we showed in Chapter 6 (Alg. 6.1) how to perform a 2-input (bits) 1-output conditional statement in R1CS. This is a constant 2-bit lookup table that costs 3C. This technique can be applied for larger window tables, but the multiplicative depth of the evaluation increases exponentially. For the $m = 1 \in \mathbb{F}_{p^{12}}$ conditional statement, we need at least a 6-bit lookup table to check that $m_1 = 0 \in \mathbb{F}_{p^6}$, making this idea not worth investigating further.

7.3 Implementation and benchmark

To the best of our knowledge, there are only two implementations of pairings in R1CS. One in `libsnark` [BSCT+b] for MNT4 and MNT6 curves and one in `arkworks` [ac22] for

the BLS12-377 curve. The first one was written in C++ language and used previously in the Mina blockchain but is now obsolete as these MNT4/6 curves are quite inefficient at the 128-bit security level. The second one is in Rust language and corresponds exactly to the problem we investigate in this chapter. It uses a BW6-761 curve to SNARK-prove an optimal ate pairing over BLS12-377 in more than 19000 constraints.

We choose to implement our work in Go using our open-source `gnark` ecosystem [BPH+22a]. We both implement a pairing over BLS12-377 in a BW6-761 SNARK circuit and a BLS24-315 in a BW6-633 SNARK circuit. For this, we make use of all ideas discussed in this paper to implement finite field arithmetic in $\mathbb{F}_{p^2}, \mathbb{F}_{p^4}, \mathbb{F}_{p^6}, \mathbb{F}_{p^{12}}$ and $\mathbb{F}_{p^{24}}$, $\mathbb{G}_1$ and $\mathbb{G}_2$ operations and optimal ate pairings on BLS12 and BLS24. Moreover, as applications, we implement and optimize circuits for Groth16 [Gro16] verification, BLS signature verification and KZG polynomial commitment opening. Tables 7.7 and 7.8 give the overall cost of these circuits in terms of number of constraints `C`, which is almost half the best previously known implementation cost.

Table 7.7: Pairing cost for BLS12-377 and BLS24-315 in R1CS.

	Miller loop	Final exponentiation	total
`arkworks` (BLS12-377)	≈ 6000`C`	≈ 13000`C`	≈ 19000`C`
`gnark` (BLS12-377)	5519`C`	6016`C`	11535`C`
`gnark` (BLS24-315)	8132`C`	19428`C`	27608`C`

Table 7.8: Pairing-based circuits costs in R1CS for BLS12-377 and BLS24-315.

	Groth16 verif.	BLS sig. verif.	KZG poly. commit.
`gnark` (BLS12-377)	19378`C`	14888`C`	20691`C`
`gnark` (BLS24-315)	40275`C`	32626`C`	57331`C`

Note that the BLS signature verification circuit excludes the hash-to-curve cost and that the KZG circuit needs a scalar multiplication in $\mathbb{G}_2$ which we implement in 3.5`C` per bit of the scalar following [BGH19, Sec. 6.2 - Alg. 1] using the $D = 3$ endomorphism ϕ in R1CS (cf. Alg. 7.3).

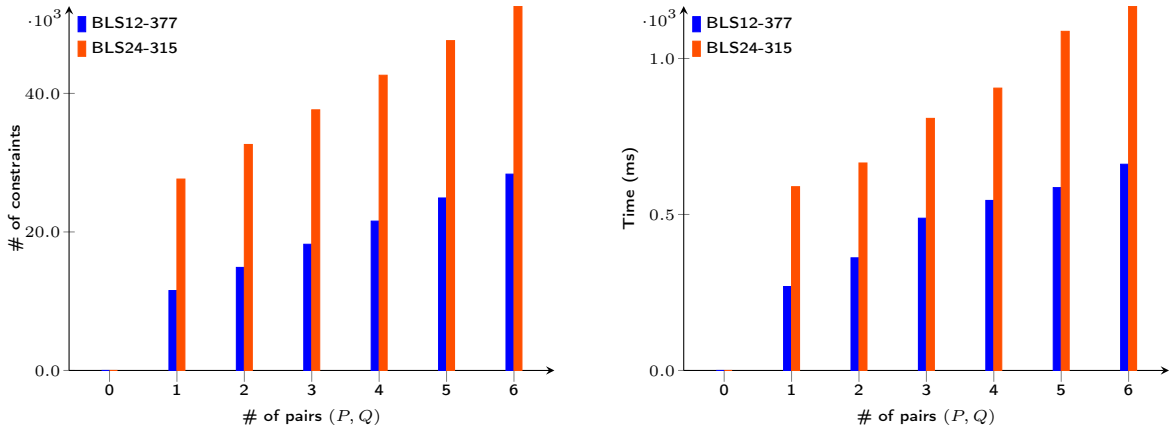
Algorithm 7.3: Endomorphism-based scalar multiplication in Weierstrass form in R1CS.
Input: E defined over $\mathbb{F}_p$, $m > 0$, $P \in E(\mathbb{F}_p) \setminus \mathcal{O}$
Output: $[m]P \in E$

Write m in binary expansion $m = \sum_{i=0}^{n-1} b_i 2^i$ where $b_i \in \{0, 1\}$
$R \leftarrow [2](P + \phi(P))$
for $i = n/2 - 1$ *dowto* 0**:**
    if $b_{2i+1} = 0$**:**
        $S_i \leftarrow [2b_{2i} - 1]P$
    else:
        $S_i \leftarrow \phi([2b_{2i} - 1]P)$
    $R \leftarrow (R + S_i) + R$
return R

Timings

The number of constraints is independent of the choice of a programming language and the usual software concerns. However, to better highlight the consequence of this work, we report in Figure 7.1 timings of the Groth16 Prove algorithm corresponding to a single pairing, multi-pairings and pairing-based circuits on a AMD EPYC 7R32 AWS (c5a.24xlarge) machine.

We use the groth16 implementation in our open-source library `gnark` [BPH+22a] where we implemented the pairing circuits for BLS12-377 and BLS24-315. We run the benchmark with hyperthreading, turbo and frequency scaling disabled.



(a) Number of constraints.

(b) Proving times.

Figure 7.1: Groth16 number of constraints (a) and proving times (b) for multi-pairings on BLS12-377 and BLS24-315 curves.

Conclusion

Summary

Business runs on information. The faster it's received and the more accurate it is, the better. Blockchain is ideal for delivering that information because it provides immediate, shared and completely transparent information stored on an immutable ledger that can be accessed only by permissioned network members. A blockchain network can track orders, payments, accounts, production and much more. And because members share a single view of the truth, you can see all details of a transaction end to end, giving you greater confidence, as well as new efficiencies and opportunities. Many have likened the revolutionary possibilities of blockchain technology to those of the internet, such is its perceived capacity to transform the ways in which people and businesses cooperate. However this technology still, to some extent, suffers from privacy and scalability issues.

How can these issues in blockchain networks be overcome? By default, blockchains allow tracing the transaction history of users. They furthermore are slow and have a low throughput, i.e., the number of transactions per second is typically less than 100. Solutions for privacy include transaction mixing and cryptographic means that hide transaction information. Scalability solutions either change the consensus algorithm of the blockchain or move transactions off-chain, i.e., provide mechanisms for conducting transactions that are not recorded on the blockchain while maintaining almost the same level of security. To address this dilemma zero-knowledge proof systems and in particular zk-SNARKs have shown to be a key solution. They provide a computationally sound proof, cheap to verify and small compared to the size of the statement or the witness. Instead of publishing an expensive computation on the blockchain, a party publishes a zero-knowledge SNARK proof of the correct execution of that computation. This solves both the privacy issue and the scalability issue.

Pairing-based SNARKs are the most efficient proof systems with respect to the succinctness property. Furthermore, this property makes them good candidates for recursive proof composition. A recursive proof is a proof made by a prover that convinces a verifier that other proofs made by other provers have been correctly verified by the prover. This would allow a single proof to inductively attest to the correctness of many former proofs, yielding even more scalable blockchains.

In this dissertation, we investigated the arithmetic of recursive pairing-based proof systems. We presented a study at three stages of the process: curves to instantiate a SNARK (Chapter 3), curves to instantiate a recursive SNARK (Chapter 4), and also curves to express an elliptic-curve related statement (Chapter 6). We provided new constructions of curves for SNARKs and new families of 2-chain curves for recursive SNARKs (Chapter 4). We derived and implemented in open-source (`gnark-crypto` [BPH+22b]) efficient algorithms to speed up the arithmetic on these curves: co-factor clearing (Chapter 3), subgroup membership testing (Chapter 3), multi-scalar multiplication (Chapter 5) and pairings over 2-chains (Chapter 4). We also studied and optimized elliptic-curve arithmetic (Chapter 6) and pairings (Chapter 7) as a SNARK statement, yielding to the fastest recursive proof generation in pairing-based settings.

Open questions

Throughout the dissertation, we littered a number of open questions. We collect them hereafter:

- Is it possible to find a silver bullet construction of elliptic curves that can address all the efficiency/security requirements?
- Are there cycles of elliptic curves with the same embedding degree, and possibly the same discriminant?
- Are there pairing-friendly cycles of embedding degrees greater than 6?
- Are there pairing-friendly cycles combining MNT, Freeman and Barreto-Naehrig curves?
- Are there more efficient cycles of pairing- friendly curves? How to generate them?
- What are the optimal choices for 2-chains at higher security levels?
- Are there more efficient constructions of 2-chains with smaller outer curves?
- Is it possible to mutualize a maximum of computations between a $\mathbb{G}_1$ MSM instance and a $\mathbb{G}_2$ MSM instance?

We hope that our dissertation attracts new interests towards this topic, and that further developments translate into practical impact through one of the many frameworks for theoretical research or practical deployment in this space.

Appendix

Appendix

A

Some open-source implementations of SNARK-friendly curves

We report in Table A.1 some libraries that implement different SNARK curves, 2-chains and 2-cycles. We only cite implementations that are used in zero-knowledge proofs based projects and we omit to cite forks that improve independently over the original work. The libraries are implemented in different languages and some use more assembly acceleration than others. Besides the different algorithmic and software optimizations used across them, it should also be noted that some libraries target constant-time implementations for some or all the operations.

Note. Libraries in Table A.1 provide the classical implementation of elliptic curves. Few of these libraries provide also implementations of curves as SNARK computations. That is, the arithmetic of fields and groups of elliptic curves as statements to be proved in a SNARK using another elliptic curve (e.g. Alg. 6.2 for twisted Edwards). For example arkworks [ac22], gnark [BPH+22a], libsnark [BSCT+b] and zcash [BS] provide such implementations within different proving systems.

[4] gnark-crypto and arkworks implement for each SNARK curve an associated twisted Edwards curve, including Jubjub and Bandersnatch.

Library	curves implemented	programming language	license	zero-knowledge projects
arkworks-rs [ac22]	BN254 BLS12-381 BLS12-377/BW6-761 chain MNT753 cycle MNT298 cycle Pasta cycle (and all associated curves[4])	Rust	MIT Apache-2.0	Celo Aleo Espresso (jellyfish)
Barretenberg [WG]	BN254	C++	GPL-2.0	Aztec rollup
blst [Sup]	BLS12-381	C	Apache-2.0	Filecoin Ethereum Foundation
constantine [And]	BN254 BLS12-381 BLS12-377 Jubjub, Bandersnatch Curve25519	Nim	MIT Apache-2.0	Status-Ethereum (ongoing adoption)
Dalek [dVYA22]	Ed25519 Curve25519	Rust	BSD-3-Clause	Dalek-bulletproofs Spartan
Geth (Cloudflare) [ged]	BN254	Go	LGPL-3.0	Geth Erigon
gnark-crypto[4] [BPH+22b]	BN254 BLS12-381 BLS24-317 BLS12-377/BW6-761 chain BLS24-315/BW6-633 chain (and all associated curves[4])	Go	Apache-2.0	gnark ConsenSys Rollup Algorand Baseline protocol Geth (Fuzzing)
Kilic [Kil]	BN254 BLS12-381 BLS12-377/BW6-761 chain	Go	Apache-2.0	Geth Celo
libff [BSCT+a]	BN254 BLS12-381 GMV6-183 MNT298 cycle	C++	MIT	Libsnark Loopring
mcl [Shi]	BN254 BLS12-381	C++	BSD-3-Clause	DFINITY
RELIC [AGM+22]	BN254 BLS12-377 BLS12-381 BLS24-315 Tweedle/Pasta cycles	C	Apache-2.0 LGPL-2.1	Chia Network
wasmcurves [Bay]	BN254 BLS12-381	JavaScript, WASM	GPL-3	Circom/snarkjs Polygon Hermez rollup
zcash [BS, Zca]	BN254 BLS12-381 Jubjub Pasta cycle	Rust	MIT Apache-2.0	Zcash Algorand zkSync rollup

Table A.1: Some open-source implementations of SNARK-friendly curves.

Bibliography

[ABLR14] Diego F. Aranha, Paulo S. L. M. Barreto, Patrick Longa, and Jefferson E. Ricardini. The realm of the pairings. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 3–25. Springer, Heidelberg, August 2014.

[ac22] arkworks contributors. `arkworks` zkSNARK ecosystem. `https://arkworks.rs`, 2022.

[AEHG22] Diego F. Aranha, Youssef El Housni, and Aurore Guillevic. A survey of elliptic curves for proof systems. Cryptology ePrint Archive, Report 2022/586, 2022. `https://eprint.iacr.org/2022/586`.

[AF07] Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 118–136. Springer, Heidelberg, February 2007.

[AFK+13] Diego F. Aranha, Laura Fuentes-Castañeda, Edward Knapp, Alfred Menezes, and Francisco Rodríguez-Henríquez. Implementing pairings at the 192-bit security level. In Michel Abdalla and Tanja Lange, editors, *PAIRING 2012*, volume 7708 of *LNCS*, pages 177–195. Springer, Heidelberg, May 2013.

[AGM+22] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`, 2022.

[AKL+11] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio Cesar López-Hernández. Faster explicit formulas for computing pairings over ordinary curves. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 48–68. Springer, Heidelberg, May 2011.

[And] Mamy André-Ratsimbazafy. Constant time pairing-based or elliptic curve based cryptography and digital signatures.

[Bay] Jordi Baylina. Web assembly low level implementation of pairing friendly curves.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCG+20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964. IEEE Computer Society Press, May 2020.

[BCKL21] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast fourier transform (ECFFT) part I: fast polynomial algorithms over all finite fields. *CoRR*, abs/2107.08473, 2021.

[BCM+15] Paulo S. L. M. Barreto, Craig Costello, Rafael Misoczki, Michael Naehrig, Geovandro C. C. F. Pereira, and Gustavo Zanon. Subgroup security in pairing-based cryptography. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 245–265. Springer, Heidelberg, August 2015.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 1–18. Springer, Heidelberg, November 2020.

[BCTV14a] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014.

[BCTV14b] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.

[BD19] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, 32(4):1298–1336, October 2019.

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 649–680, Virtual Event, August 2021. Springer, Heidelberg.

[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, September 2012.

[BDLO12] Daniel J. Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. Faster batch forgery identification. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 454–473. Springer, Heidelberg, December 2012.

[BFR+13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 341–357, New York, NY, USA, 2013. Association for Computing Machinery. ePrint with major differences at `ePrint 2013/356`.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

[BGGM15] Razvan Barbulescu, Pierrick Gaudry, Aurore Guillevic, and François Morain. Improving NFS for the discrete logarithm problem in non-prime finite fields. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 129–155. Springer, Heidelberg, April 2015.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. `https://eprint.iacr.org/2019/1021`.

[BGJT14] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 1–16. Springer, Heidelberg, May 2014.

[BGK15] Razvan Barbulescu, Pierrick Gaudry, and Thorsten Kleinjung. The tower number field sieve. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 31–55. Springer, Heidelberg, November / December 2015.

[BGM+10] Jean-Luc Beuchat, Jorge E. González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal Ate pairing over Barreto-Naehrig curves. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *PAIRING 2010*, volume 6487 of *LNCS*, pages 21–39. Springer, Heidelberg, December 2010.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, Heidelberg, February 2005.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013.

[BKLS02] Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 354–368. Springer, Heidelberg, August 2002.

[BL] Daniel J. Bernstein and Tanja Lange. `https://safecurves.cr.yp.to`, accessed 28 February 2022.

[BL22] Daniel Bernstein and Tanja Lange. Explicit-formulas database. `https://www.hyperelliptic.org/EFD/`, 2022.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[BLS03] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267. Springer, Heidelberg, September 2003.

[BLS04] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. On the selection of pairing-friendly groups. In Mitsuru Matsui and Robert J. Zuccherato, editors, *SAC 2003*, volume 3006 of *LNCS*, pages 17–25. Springer, Heidelberg, August 2004.

[BMRS20] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352, 2020. `https://eprint.iacr.org/2020/352`.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, August 2006.

[Bow17] Sean Bowe. BLS12-381: New zk-SNARK elliptic curve construction. Zcash blog, March 11 2017. https://blog.z.cash/new-snark-curve/.

[Bow19] Sean Bowe. Faster subgroup checks for BLS12-381. Cryptology ePrint Archive, Report 2019/814, 2019. https://eprint.iacr.org/2019/814.

[BP04] Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 48–62. Springer, Heidelberg, December 2004.

[BPH+22a] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark, 2022.

[BPH+22b] Gautam Botrel, Thomas Piellard, Youssef El Housni, Arya Tabaie, and Ivo Kubjas. Go library for finite fields, elliptic curves and pairings for zero-knowledge proof systems, 2022.

[Bro10] Daniel R. L. Brown. Sec 2: Recommended elliptic curve domain parameters. http://www.secg.org/sec2-v2.pdf, 2010.

[BS] Sean Bowe and Str4d. Zero-knowledge cryptography in rust.

[BSCT+a] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, Madars Virza, Howard Wu, and Contributors. C++ library for finite fields and elliptic curves.

[BSCT+b] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, Madars Virza, Howard Wu, and Contributors. C++ library for zksnark.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. https://eprint.iacr.org/2013/404.

[BW05] Friederike Brezing and Annegret Weng. Elliptic curves suitable for pairing based cryptography. *Des. Codes Cryptogr.*, 37(1):133–141, 2005.

[CBGW10] Craig Costello, Colin Boyd, Juan Manuel González Nieto, and Kenneth Koon-Ho Wong. Avoiding full extension field arithmetic in pairing computations. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 203–224. Springer, Heidelberg, May 2010.

[CCW19] Alessandro Chiesa, Lynn Chua, and Matthew Weidner. On cycles of pairing-friendly elliptic curves. *SIAM Journal on Applied Algebra and Geometry*, 3(2):175–192, 2019.

[CDS20] Rémi Clarisse, Sylvain Duquesne, and Olivier Sanders. Curves with fast computations in the first pairing group. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 280–298. Springer, Heidelberg, December 2020.

[CFH+15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy, SP*

2015, San Jose, CA, USA, May 17-21, 2015, pages 253–270. IEEE Computer Society, 2015. ePrint 2014/976.

[CH07] Jaewook Chung and M. Anwar Hasan. Asymmetric squaring formulae. In *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, pages 113–122, 2007.

[Cha06] Denis Charles. On the existence of distortion maps on ordinary elliptic curves. Cryptology ePrint Archive, Report 2006/128, 2006. `https://eprint.iacr.org/2006/128`.

[Che10] Jung Hee Cheon. Discrete logarithm problems with auxiliary inputs. *Journal of Cryptology*, 23(3):457–476, July 2010.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

[CHZ22] Shi Ping CAI, Zhi HU, and Chang An ZHAO. Faster final exponentiation on the kss18 curve. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, advpub:2021–2086, 2022.

[CL15] Craig Costello and Patrick Longa. Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the Mersenne prime. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 214–235. Springer, Heidelberg, November / December 2015.

[CLN10] Craig Costello, Tanja Lange, and Michael Naehrig. Faster pairing computations on curves with high-degree twists. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 224–242. Springer, Heidelberg, May 2010.

[DFKP16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy*, pages 235–254. IEEE Computer Society Press, May 2016.

[DGP20] Gabrielle De Micheli, Pierrick Gaudry, and Cécile Pierrot. Asymptotic complexities of discrete logarithm algorithms in pairing-relevant finite fields. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 32–61. Springer, Heidelberg, August 2020.

[DGP21] Gabrielle De Micheli, Pierrick Gaudry, and Cécile Pierrot. Lattice enumeration for tower NFS: A 521-bit discrete logarithm computation. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 67–96. Springer, Heidelberg, December 2021.

[DIM05] Vassil S. Dimitrov, Laurent Imbert, and Pradeep Kumar Mishra. Efficient and secure elliptic curve point multiplication using double-base chains. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 59–78. Springer, Heidelberg, December 2005.

[dR95] Peter de Rooij. Efficient exponentiation using procomputation and vector addition chains. In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 389–399. Springer, Heidelberg, May 1995.

[dVYA22] Henry de Valence, Cathie Yun, and Oleg Andreev. dalek cryptography: Fast, sage, pure-rust elliptic curve cryptography, 2022.

[EH] Youssef El Housni. Benchmarking pairing-friendly elliptic curves libraries. https://hackmd.io/@gnark/eccbench.

[EHG20] Youssef El Housni and Aurore Guillevic. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 259–279. Springer, Heidelberg, December 2020.

[EHG21] Youssef El Housni and Aurore Guillevic. Families of SNARK-friendly 2-chains of elliptic curves. `https://gitlab.inria.fr/zk-curves/snark-2-chains`, 10 2021. MIT License.

[EHG22] Youssef El Housni and Aurore Guillevic. Families of SNARK-friendly 2-chains of elliptic curves. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 367–396. Springer, Heidelberg, May / June 2022.

[EHGP22] Youssef El Housni, Aurore Guillevic, and Thomas Piellard. Co-factor clearing and subgroup membership testing on pairing-friendly curves. In Lejla Batina and Joan Daemen, editors, *Progress in Cryptology - AFRICACRYPT 2022*, pages 518–536, Cham, 2022. Springer Nature Switzerland.

[ELM03] Kirsten Eisenträger, Kristin Lauter, and Peter L. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 343–354. Springer, Heidelberg, April 2003.

[ES10] Andreas Enge and Andrew V. Sutherland. Class invariants by the crt method. In Guillaume Hanrot, François Morain, and Emmanuel Thomé, editors, *Algorithmic Number Theory*, pages 142–156, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[FK19] Georgios Fotiadis and Elisavet Konstantinou. TNFS resistant families of pairing-friendly elliptic curves. *Theoretical Computer Science*, 800:73–89, 31 December 2019.

[FNK12] Fujitsu Laboratories, NICT, and Kyushu University. DL record in $\mathbb{F}_{3^{6\cdot 97}}$ of 923 bits (278 dd). NICT press release, June 18, 2012. `http://www.nict.go.jp/en/press/2012/06/18en-1.html`.

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 44–61. Springer, Heidelberg, May / June 2010.

[FST10] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, April 2010.

[Gab19] Ariel Gabizon. AuroraLight: Improved prover efficiency and SRS size in a sonic-like system. Cryptology ePrint Archive, Report 2019/601, 2019. `https://eprint.iacr.org/2019/601`.

[GAL+13] Gurleen Grewal, Reza Azarderakhsh, Patrick Longa, Shi Hu, and David Jao. Efficient implementation of bilinear pairings on ARM processors. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 149–165. Springer, Heidelberg, August 2013.

[ged] go-ethereum developers. Official go implementation of the ethereum protocol.

[GF16] Loubna Ghammam and Emmanuel Fouotsa. On the computation of the optimal ate pairing at the 192-bit security level. Cryptology ePrint Archive, Report 2016/130, 2016. `https://eprint.iacr.org/2016/130`.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GKL+21] Robert Granger, Thorsten Kleinjung, Arjen K. Lenstra, Benjamin Wesolowski, and Jens ZumbrÃ€gel. Computation of a 30750-bit binary field discrete logarithm. *Math. Comp.*, 90(332):2997–3022, 2021. ePrint `2020/965`.

[GKM+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security Symposium*, 2021.

[GKZ14] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. Breaking '128-bit secure' supersingular binary curves - (or how to solve discrete logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$). In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 126–145. Springer, Heidelberg, August 2014.

[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, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 518–535. Springer, Heidelberg, April 2009.

[GLV01] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 190–200. Springer, Heidelberg, August 2001.

[GM16] Aurore Guillevic and François Morain. *Pairings for Engineers*, chapter 9 – Discrete Logarithms, pages 203–242. CRC Press Taylor and Francis group, Spring 2016. N. ElMrabet and M. Joye (eds), `https://www.crcpress.com/Guide-to-Pairing-Based-Cryptography/El-Mrabet-Joye/p/book/9781498729505` `https://hal.inria.fr/hal-01420485v2`.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMT20] Aurore Guillevic, Simon Masson, and Emmanuel Thomé. Cocks–Pinch curves of embedding degrees five to eight and optimal ate pairing computation. *Des. Codes Cryptogr.*, 88:1047–1081, March 2020.

[GMV04] Steven D. Galbraith, J. McKee, and P. Valenca. Ordinary abelian varieties having small embedding degree. Cryptology ePrint Archive, Report 2004/365, 2004. `https://eprint.iacr.org/2004/365`.

[GMV07] Steven D. Galbraith, James F. McKee, and P. C. Valença. Ordinary abelian varieties having small embedding degree. *Finite Fields Their Appl.*, 13(4):800–814, 2007.

[gna20] gnark. Faster big-integer modular multiplication for most moduli. `https://hackmd.io/@gnark/modular_multiplication`, 2020.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, August 2006.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Heidelberg, December 2006.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[GS08a] Steven D. Galbraith and Michael Scott. Exponentiation in pairing-friendly groups using homomorphisms. In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 211–224. Springer, Heidelberg, September 2008.

[GS08b] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

[GS10] Robert Granger and Michael Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 209–223. Springer, Heidelberg, May 2010.

[GS21] Aurore Guillevic and Shashank Singh. On the alpha value of polynomials in the tower number field sieve algorithm. *Mathematical Cryptology*, 1(1), Feb. 2021.

[Gui20] Aurore Guillevic. A short-list of pairing-friendly curves resistant to special TNFS at the 128-bit security level. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 535–564. Springer, Heidelberg, May 2020.

[Gui21] Aurore Guillevic. Pairing-friendly curves. `https://members.loria.fr/AGuillevic/pairing-friendly-curves/`, 2021.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[GW20] Ariel Gabizon and Zachary Williamson. Proposal: The turbo-plonk program syntax for specifying snark programs. `https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf`, 2020.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

[Ham15] Mike Hamburg. Decaf: Eliminating cofactors through point compression. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 705–723. Springer, Heidelberg, August 2015.

[HB22] Youssef EL Housni and Gautam Botrel. Edmsm: Multi-scalar-multiplication for recursive snarks and more. Cryptology ePrint Archive, Paper 2022/1400, 2022. `https://eprint.iacr.org/2022/1400`.

[HHT20] Daiki Hayashida, Kenichiro Hayasaka, and Tadanori Teruya. Efficient final exponentiation via cyclotomic structure for pairings over families of elliptic curves. Cryptology ePrint Archive, Report 2020/875, 2020. `https://eprint.iacr.org/2020/875`.

[Hop20] Daira Hopwood. The pasta curves for halo 2 and beyond. `https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/`, 2020.

[Hop21] Daira Hopwood. Pluto-eris hybrid cycle of elliptic curves, 2021. `https://github.com/daira/pluto-eris`.

[Hou22] Youssef El Housni. Pairings in rank-1 constraint systems. Cryptology ePrint Archive, Report 2022/1162, 2022. https://eprint.iacr.org/2022/1162.

[HSV06] Florian Hess, Nigel P. Smart, and Frederik Vercauteren. The eta pairing revisited. *IEEE Trans. Inf. Theory*, 52(10):4595–4602, 2006.

[HWCD08] Hüseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted Edwards curves revisited. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 326–343. Springer, Heidelberg, December 2008.

[JKS16] Ari Juels, Ahmed E. Kosba, and Elaine Shi. The ring of Gyges: Investigating the future of criminal smart contracts. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 283–295. ACM Press, October 2016.

[Jon13] Nathan Jones. Elliptic aliquot cycles of fixed length. *Pacific Journal of Mathematics*, 263(2):353–371, 2013.

[JP14] Antoine Joux and Cécile Pierrot. The special number field sieve in $\mathbb{F}_{p^n}$ - application to pairing-friendly constructions. In Zhenfu Cao and Fangguo Zhang, editors, *PAIRING 2013*, volume 8365 of *LNCS*, pages 45–61. Springer, Heidelberg, November 2014.

[Kar13] Koray Karabina. Squaring in cyclotomic subgroups. *Math. Comput.*, 82(281):555–579, 2013.

[KB16] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 543–571. Springer, Heidelberg, August 2016.

[Kil] Onur Kilic. High-speed implementation of curves in go. https://github.com/kilic/bn254, https://github.com/kilic/bls12-381, https://github.com/kilic/bls12-377 and https://github.com/kilic/bw6.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[KMS+16] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858. IEEE Computer Society Press, May 2016.

[KO63] A. Karatsuba and Yu. Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics Doklady*, 7:595, January 1963.

[Kos21] Dmitrii Koshelev. Batch point compression in the context of advanced pairing-based protocols. Cryptology ePrint Archive, Report 2021/1446, 2021. https://eprint.iacr.org/2021/1446.

[KPP+14] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. TRUESET: Faster verifiable set computations. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 765–780. USENIX Association, August 2014.

[KPV19] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400, 2019. https://eprint.iacr.org/2019/1400.

[KSS08] Ezekiel J. Kachisa, Edward F. Schaefer, and Michael Scott. Constructing Brezing-Weng pairing-friendly elliptic curves using elements in the cyclotomic field. In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 126–135. Springer, Heidelberg, September 2008.

[KT08] Koray Karabina and Edlyn Teske. On prime-order elliptic curves with embedding degrees k = 3, 4, and 6. In Alfred J. van der Poorten and Andreas Stein, editors, *Algorithmic Number Theory*, pages 102–117, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[KW22] Thorsten Kleinjung and Benjamin Wesolowski. Discrete logarithms in quasi-polynomial time in finite fields of fixed characteristic. *J. Amer. Math. Soc.*, 35(02):581–624, Jan 2022. ePrint 2019/751.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[KZM+15] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015. https://eprint.iacr.org/2015/1093.

[LMHT18] Duc-Phong Le, Nadia El Mrabet, Safia Haloui, and Chik How Tan. On the near prime-order MNT curves. *CoRR*, abs/1806.02536, 2018.

[Lyn07] Benjamin Lynn. *On the implementation of pairing-based cryptosystems*. Phd thesis, Stanford University, department of computer science, 2007. https://crypto.stanford.edu/pbc/thesis.html.

[Lyn13] Ben Lynn. Pairing-based cryptography (PBC) library. https://crypto.stanford.edu/pbc/, 2013. v-0.5.14. C language, LGPL license.

[Mas20] Simon Masson. *Algorithmic of curves in the context of bilinear and post-quantum cryptography*. Doctorat, Université de Lorraine, Nancy, France, December 2020. https://tel.archives-ouvertes.fr/tel-03052499.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

[Mec20] Izaak Meckler. O(1) labs fork of zexe: implementation of bn382-plain, 2020. https://github.com/o1-labs/zexe/tree/master/algebra/src/bn_382.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

[Mih07a] Preda Mihailescu. Dual elliptic primes and applications to cyclotomy primality proving, 2007.

[Mih07b] Preda Mihailescu. Dual elliptic primes and applications to cyclotomy primality proving. arXiv `0709.4113`, 2007.

[MMV+14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In Antoine Joux and Amr M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 306–323. Springer, Heidelberg, August 2014.

[MNT01] Atsuko Miyaji, Masaki Nakabayashi, and Shunzo Takano. Characterization of elliptic curve traces under FR-reduction. In Dongho Won, editor, *ICISC 00*, volume 2015 of *LNCS*, pages 90–108. Springer, Heidelberg, December 2001.

[MO90] François Morain and Jorge Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 24(6):531–543, 1990.

[Mon87] Peter L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.

[Mor07] François Morain. The ecpp home page, 2007. `http://www.lix.polytechnique.fr/~morain/Prgms/ecpp.english.html`.

[MSS16] Alfred Menezes, Palash Sarkar, and Shashank Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In Raphael C.-W. Phan and Moti Yung, editors, *Mycrypt Conference*, volume 10311 of *LNCS*, pages 83–108, Kuala Lumpur, Malaysia, December 1-2 2016. Springer. `https://ia.cr/2016/1102`.

[MSZ21] Simon Masson, Antonio Sanso, and Zhenfei Zhang. Bandersnatch: a fast elliptic curve built over the BLS12-381 scalar field. Cryptology ePrint Archive, Report 2021/1152, 2021. `https://eprint.iacr.org/2021/1152`.

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Heidelberg, August 2003.

[NAS+08] Yasuyuki Nogami, Masataka Akane, Yumi Sakemi, Hidehiro Katou, and Yoshitaka Morikawa. Integer variable chi-based Ate pairing. In Steven D. Galbraith and Kenneth G. Paterson, editors, *PAIRING 2008*, volume 5209 of *LNCS*, pages 178–191. Springer, Heidelberg, September 2008.

[NBS08] Michael Naehrig, Paulo S. L. M. Barreto, and Peter Schwabe. On compressible pairings and their computation. In Serge Vaudenay, editor, *AFRICACRYPT 08*, volume 5023 of *LNCS*, pages 371–388. Springer, Heidelberg, June 2008.

[NNS10] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *LATINCRYPT 2010*, volume 6212 of *LNCS*, pages 109–123. Springer, Heidelberg, August 2010.

[Par19] James Parks. An asymptotic for the average number of amicable pairs for elliptic curves. *Mathematical Proceedings of the Cambridge Philosophical Society*, 166(1):33–59, 2019.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[Pip76] Nicholas Pippenger. On the evaluation of powers and related problems (preliminary version). In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 258–263. IEEE Computer Society, 1976.

[Poe18] Andrew Poelstra. Curve with group order $2^{255} - 19$. `https://moderncrypto.org/mail-archive/curves/2018/000992.html`, accessed 28 February 2022, 2018.

[Pol71] J. M. Pollard. The Fast Fourier Transform in a finite field. *Math. Comp.*, 25(114):365–374, April 1971.

[Por20] Thomas Pornin. Optimized binary GCD for modular inversion. Cryptology ePrint Archive, Report 2020/972, 2020. `https://eprint.iacr.org/2020/972`.

[RS03] Karl Rubin and Alice Silverberg. Torus-based cryptography. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 349–365. Springer, Heidelberg, August 2003.

[Sch87] René Schoof. Nonsingular plane cubic curves over finite fields. *Journal of Combinatorial Theory, Series A*, 46(2):183–211, 1987.

[Sco09] Michael Scott. A note on twists for pairing friendly curves, 2009.

[Sco13] Michael Scott. Unbalancing pairing-based key exchange protocols. Cryptology ePrint Archive, Report 2013/688, 2013. `https://eprint.iacr.org/2013/688`.

[Sco19] Michael Scott. Pairing implementation revisited. Cryptology ePrint Archive, Report 2019/077, 2019. `https://eprint.iacr.org/2019/077`.

[Sco21] Michael Scott. A note on group membership tests for G_1, G_2 and G_T on BLS pairing-friendly curves. Cryptology ePrint Archive, Report 2021/1130, 2021. `https://eprint.iacr.org/2021/1130`.

[SG18] Michael Scott and Aurore Guillevic. A new family of pairing-friendly elliptic curves. Cryptology ePrint Archive, Report 2018/193, 2018. `https://eprint.iacr.org/2018/193`.

[Shi] MITSUNARI Shigeo. a portable and fast pairing-based cryptography library.

[SHI+12] Yumi Sakemi, Goichiro Hanaoka, Tetsuya Izu, Masahiko Takenaka, and Masaya Yasuda. Solving a discrete logarithm problem with auxiliary input on a 160-bit elliptic curve. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 595–608. Springer, Heidelberg, May 2012.

[Sil09] Joseph H Silverman. *The Arithmetic of Elliptic Curves*. Graduate texts in mathematics. Springer, Dordrecht, 2009.

[Sma99] Nigel P. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12(3):193–196, June 1999.

[SS11] Joseph H. Silverman and Katherine E. Stange. Amicable Pairs and Aliquot Cycles for Elliptic Curves. *Experimental Mathematics*, 20(3):329 – 357, 2011.

[Sta21] Martijn Stam. XTR and tori. Cryptology ePrint Archive, Report 2021/1659, 2021. `https://eprint.iacr.org/2021/1659`.

[Str64] Ernst G. Strauss. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70(114):806–808, 1964.

[Sup] Supranational. Multilingual bls12-381 signature library.

[Sut11] Andrew V. Sutherland. Computing hilbert class polynomials with the chinese remainder theorem. *Math. Comp.*, 80(273):501–538, 2011. arXiv 0903.2785.

[Tib14] Mehdi Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 139–156. Springer, Heidelberg, March 2014.

[Val21] Henry De Valence. The ristretto group. `https://ristretto.group`, 2021.

[Ver10] F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, Jan 2010.

[VP19] Alexander Vlasov and Konstantin Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. Cryptology ePrint Archive, Report 2019/1020, 2019. `https://eprint.iacr.org/2019/1020`.

[WB19] Riad S. Wahby and Dan Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR TCHES*, 2019(4):154–179, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/8348`.

[WG] Zachary Williamson and Ariel Gabizon. An optimized elliptic curve library for the bn128 curve, and plonk snark prover.

[WTs+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

[Wui21] Peter Wuille. Elligator Squared for BN-like curves. https://github.com/sipa/writeups/tree/main/elligator-square-for-bn, 2021.

[Zca] Zcash. Rust implementation for the pasta cycle in rust.

[ZCa21] ZCash. What is jubjub? https://z.cash/technology/jubjub/, 2021.

ECOLE
DOCTORALE

Titre : L'arithmétique des systèmes de preuves basés sur les couplages

Mots clés : Systèmes de preuves, SNARKs, Courbes elliptiques, Couplages bilinéaires

Résumé : Un système de preuve est un protocole où une partie (appelée le prouveur) essaie de convaincre une autre partie (appelée le vérifieur) qu'un énoncé donné est vrai. Dans la classe des systèmes de preuve non interactifs, un concept intéressant pour prouver l'intégrité de calcul est le "Succinct Non-interactive ARgument of Knowledge" (SNARK). Il fournit une preuve calculatoirement consistante, peu coûteuse à vérifier et petite de taille par rapport à la taille de l'énoncé ou du témoin. Les couplages bilinéaires sur des courbes elliptiques sont devenus des ingrédients clés pour instancier de tels SNARKs.
Dans cette thèse nous étudions des courbes elliptiques à couplage efficace adaptées à ce type de SNARKs. Nous présentons une étude à trois étapes du processus : Des courbes pour instancier un SNARK, des courbes pour instancier un SNARK récursif, et également des courbes pour exprimer un énoncé lié à l'arithmétique sur la courbe elliptique. Nous fournissons de nouvelles constructions de courbes pour les SNARKs et de nouvelles familles de 2-chaînes de courbes pour les SNARKs récursifs. Nous dérivons et implémentons en open-source des algorithmes efficaces pour accélérer l'arithmétique sur ces courbes : Effacement des cofacteurs, test d'appartenance aux sous-groupes, multiplication multi-scalaire et couplage sur les 2-chaînes. Nous étudions et optimisons également l'arithmétique des courbes elliptiques et le couplage bilinéaire en tant qu'énoncés SNARK à prouver, permettant de générer rapidement une preuve récursive.

Title : The arithmetic of pairing-based proof systems

Keywords : Proof systems, SNARKs, Elliptic curves, Bilinear pairings

Abstract : A proof system is a protocol where one party (called the prover) tries to convince another party (called the verifier) that a given statement is true. In the class of non-interactive proof systems, a particularly interesting concept for proving the computational integrity is the Succinct Non-interactive ARgument of Knowledge (SNARK). It provides a computationally sound proof, cheap to verify and small compared to the size of the statement or the witness. Bilinear pairings over elliptic curves have become key ingredients for instantiating such SNARKs.
In this thesis we investigate tailored pairing-friendly elliptic curves to efficiently implement SNARKs. We present a study at three stages of the process : curves to instantiate a SNARK, curves to instantiate a recursive SNARK, and also curves to express an elliptic-curve related statement. We provide new constructions of curves for SNARKs and new families of 2-chain curves for recursive SNARKs. We derive and implement in open-source efficient algorithms to speed up the arithmetic on these curves : Co-factor clearing, subgroup membership testing, multi-scalar multiplication and pairings over 2-chains. We also study and optimize elliptic curves arithmetic and pairings as a SNARK statement, yielding to the fastest recursive proof generation in pairing-based settings.

Institut Polytechnique de Paris
91120 Palaiseau, France