Pairings in Rank-1 Constraint Systems

Youssef El Housni

Pairings in Rank-1 Constraint Systems

Youssef El Housni[1,2,3][0000−0003−2873−3479]

[1] ConsenSys R&D, Paris, France
[2] LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris
[3] Inria
youssef.elhousni@consensys.net

Abstract. Bilinear pairings have been used in different cryptographic applications and demonstrated to be a key building block for a plethora of constructions. In particular, some Succinct Non-interactive ARguments of Knowledge (SNARKs) have very short proofs and very fast verification thanks to a multi-pairing computation. This succinctness makes pairing-based SNARKs suitable for proof recursion, that is proofs verifying other proofs. In this scenario one requires to express efficiently a multi-pairing computation as a SNARK arithmetic circuit. Other compelling applications such as verifying Boneh–Lynn–Shacham (BLS) signatures or Kate–Zaverucha–Goldberg (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 work, we consider efficiently implementing pairings in Rank-1 Constraint Systems (R1CS), a widely used model to express SNARK statements. We implement our techniques in the `gnark` open-source ecosystem and show that the arithmetic circuit depth can be almost halved compared to the previously best known pairing implementation on a Barreto–Lynn–Scott (BLS) curve of embedding degree 12, resulting in a significantly faster proving time. We also investigate and implement the case of BLS curves of embedding degree 24.

1 Introduction

A SNARK is a cryptographic primitive that enables a prover (Alice) to prove to a verifier (Bob) the knowledge of a satisfying witness to a Non-deterministic Polynomial (NP) statement by producing a proof π such that the size of π and the cost to verify it are both sub-linear in the size of the witness. If π does not reveal anything about the witness we refer to the cryptographic primitive as a zero-knowledge (zk) SNARK.

Building on ideas from the pairing-based doubly-homomorphic encryption scheme [8], Groth, Ostrovsky and Sahai [26] introduced the pairing-based non-interactive zero-knowledge proofs, yielding the first linear-size proofs based on

standard assumptions. Groth [23] 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. Follow-up works improved on these techniques leading to the most succinct and widely implemented pairing-based SNARK [24]. This, however, comes with the drawback of a statement-specific setup.

More recently, a new kind of SNARKs was introduced, where the setup is not specific to a given statement but is rather universal in that sense. Groth et al. [25] proposed a universal SNARK with a single setup to prove all statements of a given bounded size. Sonic [35] built on that to construct the first practical universal SNARK. This work inspired many researchers and practitioners who then came up with new and elegant universal constructions such as PLONK [19] and Marlin [13]. A key building block of these universal constructions is the use of polynomial commitment (PC) schemes. While there are different PC schemes with trade-offs, the pairing-based Kate–Zaverucha–Goldberg (KZG) scheme [31] remains the most efficient.

By exploiting their succinctness, both these constructions are good candidates for recursive proof composition. Such proofs could themselves verify the correctness of (a batch of) other proofs. To this end, one should express the verification algorithm as a new SNARK statement to prove. Both Groth16 and KZG-based universal SNARKs rely on multi-pairing computations to verify a proof. That is, one should efficiently write a pairing as a SNARK circuit. Other applications fall into the same problem and motivate further this work. For example, Celo blockchain needs to generate a Groth16 proof that verifies a BLS signature [9] which is also a multi-pairings equation. This is already used in production and this work would allow to significantly speedup the proof generation. Another example is the decentralized private computation (DPC) as introduced in ZEXE [11]. This is used by the Aleo and Espresso systems [44] and could benefit directly from this work. A last example is the zk-rollup which is an active area of research and development within the Ethereum blockchain community. It aims at solving the platform scalability problem by proving a batch of transactions and only submitting the proof to the consensus layer. A promising line of work is the zk-EVM rollup (e.g. a specification by ConsenSys [33]) uses a KZG-based scheme to prove the Ethereum Virtual Machine (EVM) correct execution. This requires proving pairing computations and this work would increase the number of transactions that can fit in a zkEVM circuit.

While the traditional implementation of pairings was thoroughly considered in the literature, very little work and no previous publication has addressed the question of efficiently implementing a pairing as a SNARK arithmetic circuit. In this work, we consider efficiently implementing pairings in Rank-1 Constraint Systems (R1CS), a widely used model to express SNARK statements.

Organization. Section 2 provides some preliminaries on pairing-based SNARKs and rank-1 constraint systems. Sections 3, 5 and 4 lay out mathematical results on bilinear pairings, pairing-friendly 2-chains and algebraic tori. The contributions of the paper are Sections 6 and 7. First, we investigate efficient techniques

to express a pairing in R1CS and next we provide an optimized implementation in the context of the Groth16 [24] proof system. Finally, we discuss relevant applications and future work.

2 Pairing-based SNARKs

In the following, we mainly focus on preprocessing 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. 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 trapdoored an additional secret input τ is required $(\sigma_p, \sigma_v) \leftarrow \texttt{Setup}(F, \tau, 1^\lambda)$.

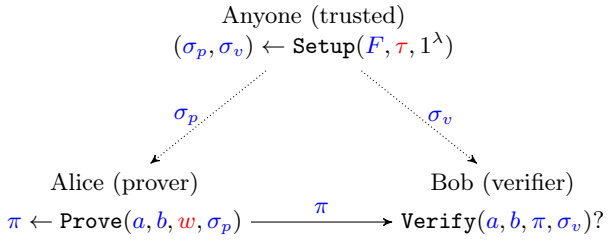


Fig. 1. zk-SNARK algorithms. Public parameters are in blue and private ones in red.

Two pairing-based schemes are particularly widely implemented in different projects. Groth16 [24] using a circuit-specific setup and PLONK [19] using a universal setup for the KZG polynomial commitment. Table 1 gives the cost of `Setup`, `Prove` and `Verify` for these two schemes.

2.1 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

Table 1. Cost of `Setup`, `Prove` and `Verify` algorithms for [24] 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. `FFT`=Fast Fourier Transform.

	`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 $\mathtt{M}_{\mathbb{G}_2}$ 7 `FFT`	3 `P` ℓ $\mathtt{M}_{\mathbb{G}_1}$
PLONK (KZG)	$d_{\geq n+a}$ $\mathtt{M}_{\mathbb{G}_1}$ 1 $\mathtt{M}_{\mathbb{G}_2}$ 8 `FFT`	$9(n+a)$ $\mathtt{M}_{\mathbb{G}_1}$ 8 `FFT`	2 `P` 18 $\mathtt{M}_{\mathbb{G}_1}$

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 with two incoming wires and one outcoming wire (cf. Fig. 2 for an example).

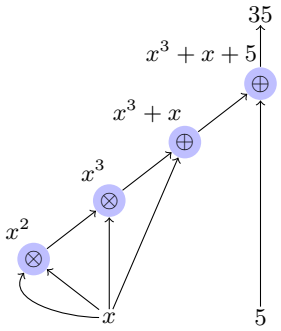


Fig. 2. Arithmetic circuit encoding the computation $x^3 + x + 5 = 35$ for which the (secret) solution is $x = 3$.

R1CS. SNARKs, such as [24], express these arithmetic circuits as a set of quadratic constraints called Rank-1 Constraint system (R1CS). It consists of two set of constraints: 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 and the internal inputs and outputs of the multiplication gates. Each multiplication gate takes two inputs and outputs their multiplication. That

relation for n gates is represented as

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

where $\vec{a_L}$ is the vector of the first (left) input to each gate, $\vec{a_R}$ the vector of the second (right) input to each gate and $\vec{a_O}$ the vector of the output. Linear constraints are expressed using a vector of equations that use linear combinations of the variables as

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

where $\vec{W_L}$, $\vec{W_R}$ and $\vec{W_O}$ are weights applied to the respective inputs and outputs of the internal variables, $\vec{W_v}$ are weights 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 prove as R1CS where the variables are in $\mathbb{F}$, a field 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 size of these variables and particularly the multiplication gates variables is what determines the prover complexity. For example, Groth16 prover complexity is dominated by the multi-scalar-multiplications (in $\mathbb{G}_1$ and $\mathbb{G}_2$) of sizes 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 [32], 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 and
- the verification can be sometimes simpler than 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. For pairing-based proof recursion we need to implement efficiently pairings in the R1CS model.

3 Background on pairings

We briefly 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 elliptic curve defined over a field $\mathbb{F}_p$, where p is a prime power. Let π_p be the Frobenius endomorphism: $(x, y) \mapsto (x^p, y^p)$. Its minimal polynomial is $X^2 - tX + p$ where t 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 $E[r] \coloneqq \{P \in E(\overline{\mathbb{F}_p}), [r]P = \mathcal{O}\}$ 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 [36], 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. The Tate and ate pairings are

Algorithm 1: MillerLoop(s, P, Q)

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

1 $m \leftarrow 1$; $S \leftarrow Q$;
2 **for** b *from* $t-1$ *to* 0 **do**
3 $\quad \ell \leftarrow \ell_{S,S}(P)$; $S \leftarrow [2]S$; // DoubleLine
4 $\quad v \leftarrow v_{[2]S}(P)$; // VerticalLine
5 $\quad m \leftarrow m^2 \cdot \ell / v$; // Update1
6 $\quad$ **if** $s_b = 1$ **then**
7 $\quad\quad \ell \leftarrow \ell_{S,Q}(P)$; $S \leftarrow S + Q$; // AddLine
8 $\quad\quad v \leftarrow v_{S+Q}(P)$; // VerticalLine
9 $\quad\quad m \leftarrow m \cdot \ell / v$; // Update2
10 **return** m;

defined by

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

where $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$. The final exponentiation kills any element which lives in a strict subfield of $\mathbb{F}_{p^k}$ [5]. 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 eliminated by the final exponentiation. Hence, in this situation we ignore the VerticalLine steps and remove the divisions by v in Update1 and Update2 steps.

It is also important to recall some results with respect to the complex multiplication (CM) equation $4p = t^2 + Dy^2$ with discriminant $-D$ and some integer y. When $-D = -3$, the curve has CM by $\mathbb{Q}(\omega)$ where $\omega^2 + \omega + 1 = 0$. In this case, a twist of degree 6 exists. It has a j-invariant 0 and is of the form $Y^2 = X^3 + b\,(a = 0)$. When E has d-th order twist for some $d \mid k$, then $\mathbb{G}_2$ is isomorphic to $E'[r](\mathbb{F}_{p^{k/d}})$ for some twist E' in this case of the form $Y^2 = X^3 + b'$. We denote by ψ 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 with $b' = b/\nu$ (D-type twist) or $b' = b\nu$ (M-type twist). For the D-type, $\psi : E' \to E : (x, y) \mapsto (\nu^{1/3}x, \nu^{1/2}y)$. For the M-type, $\psi : E' \to E : (x, y) \mapsto (\nu^{2/3}x/\nu, \nu^{1/2}y/\nu)$. For other d-twisting ψ formulas, see [41].

4 Pairing-friendly 2-chains

Following [18], a SNARK-friendly 2-chain of elliptic curves is a set of two curves as in Definition 1.

Definition 1. *A 2-chain of elliptic curves is a list of two distinct curves $E_1/\mathbb{F}_{p_1}$ and $E_1/\mathbb{F}_{p_2}$ where p_1 and p_2 are large primes and $p_1 = r_2 \mid \#E_2(\mathbb{F}_{p_2})$. Both curves should:*

- *be pairing-friendly and*
- *have a highly 2-adic subgroup, i.e. $r_1 \equiv r_2 \equiv 1 \mod 2^L$ for a large $L \geq 1$.*

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* (cf. Fig. 3).

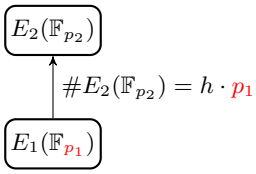


Fig. 3. A 2-chain of elliptic curves.

Inner curves from polynomial families. The best pairing-friendly elliptic curve amenable to efficient pairing implementations arise from polynomial

based families. These curves are obtained by parameterizing the CM equation with polynomials $p(x), t(x), r(x)$ and $y(x)$. The authors of [18] showed that the polynomial-based pairing-friendly Barreto–Lynn–Scott families of embedding degrees $k = 12$ (BLS12) and $k = 24$ (BLS24) [6] are the most suitable to construct inner curves in the context of pairing-based SNARKs. They showed that these curves are always of the form $E(\mathbb{F}_p) : Y^2 = X^3 + 1$ and requiring the seed x to satisfy $x \equiv 1 \bmod 3 \cdot 2^L$ is sufficient to have the 2-adicity requirement with respect to both r and p. These curves have $-D = -3$ (cf. Sec. 3).

Outer curves with the Brezing–Weng and Cocks-Pinch methods. The papers [29,18] consider 2-chains from a BLS12 and a BLS24 curve. The authors describe a general framework for all 2-chains made with a Brezing–Weng curve of embedding degree 6 (BW6) from a BLS12 curve and resp. all 2-chains of a BW6 curve from a BLS24 curve. In the sequel, we focus on efficiently proving a pairing over BLS12 and BLS24 curves in a SNARK instantiated with a BW6.

Pairings over inner BLS12 and BLS24 curves. Table 2 summarizes the salient parameters of BLS12 and BLS24 curves and Table 3 gives the concrete parameters of the curves suggested in [18] and their security, namely the BLS12-377 and BLS24-315. Next we will focus on efficient Miller loop computation and

Table 2. 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 3. Security level estimates of BLS12-377 and BLS24-315 curves from [11,18], 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	[11]	253	377	4521	2^{126}
BLS24-315, x_{315}	24	-3	[18, Tab. 10]	253	315	7543	2^{160}

final exponentiation from the literature for the case of BLS curves. The most efficient pairing on BLS curves is the optimal ate pairing [43]. 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) 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 since they are in a proper subgroup when the embedding degree k is even. These operations are best optimized following [1] for a single pairing and [40] 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 [**?**] and Chung–Hasan formulas [14]. 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 subfield, 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 [42] (instead of Lucas or $\mathbb{T}_2$ [42]). 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 subfield needed to compress $\mathbb{G}_2$ coordinates for BLS24.

$\mathbb{G}_2$ arithmetic and line evaluations. It was shown in [16,4,22,1] 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 [1], 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

$$X_{[2]S}=X_SY_S(Y_S^2-9b'Z_S^2)/2;\ Y_{[2]S}=((Y_S^2+9b'Z_S^2)/2)^2-27b'Z_S^4;\ Z_{[2]S}=2Y_S^3Z_S\ .$$

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

$$g_{[2]\psi(S)}(P)=-2Y_SZ_S\cdot y_P+3X_S^2\cdot x_Pw+(3b'Z_S^2-Y_S^2)w^3\ .$$

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 [40] 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 [28,2,20]. It is usually divided into an easy part and a hard part, as follows:

$$\begin{aligned} \frac{p^k - 1}{r} &= \underbrace{\frac{p^k - 1}{\Phi_k(p)}}_{\text{easy part}} \cdot \underbrace{\frac{\Phi_k(p)}{r}}_{\text{hard part}} \\ &= \underbrace{(p^d - 1)\frac{\sum_{i=0}^{e-1} p^{id}}{\Phi_k(p)}}_{\text{easy part}} \cdot \underbrace{\frac{\Phi_k(p)}{r}}_{\text{hard part}} \end{aligned} \tag{1}$$

where Φ_k is the k-th cyclotomic polynomial and $k = d \cdot e$. For BLS12 and BLS24 curves, the easy part is $(p^{k/2} - 1)(p^{k/d} + 1)$. 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 [28], which we suggest to implement as in Alg. 2 and Alg. 3 ($3 \cdot \Phi_k(p)/r$).

Algorithm 2: Final exp. hard part for BLS12 curves.

Input: $m = f_{x,Q}(P) \in \mathbb{F}_{p^{12}}$
Output: $m^{3\cdot\Phi_{12}(p)/r} \in \mathbb{G}_T$

1 $t_0 \leftarrow m^2$
2 $t_1 \leftarrow m^x$ // exp. to the fixed seed x
3 $t_2 \leftarrow \bar{m}$ // conjugate
4 $t_1 \leftarrow t_1 \cdot t_2$
5 $t_2 \leftarrow t_1^x$
6 $t_1 \leftarrow \bar{t_1}$
7 $t_1 \leftarrow t_1 \cdot t_2$
8 $t_2 \leftarrow t_1^x$
9 $t_1 \leftarrow t_1^p$ // Frob.
10 $t_1 \leftarrow t_1 \cdot t_2$
11 $m \leftarrow m \cdot t_0$
12 $t_0 \leftarrow t_1^x$
13 $t_2 \leftarrow t_0^x$
14 $t_0 \leftarrow t_1^{p^2}$ // Frob. square
15 $t_1 \leftarrow \bar{t_1}$
16 $t_1 \leftarrow t_1 \cdot t_2$
17 $t_1 \leftarrow t_1 \cdot t_0$
18 $m \leftarrow m \cdot t_1$
19 **return** m

Algorithm 3: Final exp. hard part for BLS24 curves.

Input: $m = f_{x,Q}(P) \in \mathbb{F}_{p^{24}}$
Output: $m^{3\cdot\Phi_{24}(p)/r} \in \mathbb{G}_T$

1 $t_0 \leftarrow m^2$
2 $t_1 \leftarrow m^x$ // exp. to the fixed seed x
3 $t_2 \leftarrow \bar{m}$ // conjugate
4 $t_1 \leftarrow t_1 \cdot t_2$
5 $t_2 \leftarrow t_1^x$
6 $t_1 \leftarrow \bar{t_1}$
7 $t_1 \leftarrow t_1 \cdot t_2$
8 $t_2 \leftarrow t_1^x$
9 $t_1 \leftarrow t_1^p$ // Frob.
10 $t_1 \leftarrow t_1 \cdot t_2$
11 $m \leftarrow m \cdot t_0$
12 $t_0 \leftarrow t_1^x$
13 $t_2 \leftarrow t_0^x$
14 $t_0 \leftarrow t_1^{p^2}$ // Frob. square
15 $t_2 \leftarrow t_0 \cdot t_2$
16 $t_1 \leftarrow t_2^x$
17 $t_1 \leftarrow t_1^x$
18 $t_1 \leftarrow t_1^x$
19 $t_1 \leftarrow t_1^x$
20 $t_0 \leftarrow t_2^{p^4}$ // Frob. quad
21 $t_0 \leftarrow t_0 \cdot t_1$
22 $t_2 \leftarrow \bar{t_2}$
23 $t_0 \leftarrow t_0 \cdot t_2$
24 $m \leftarrow m \cdot t_0$
25 **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 [21]. 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 [20], do not apply to inner BLS because the seed is always odd ($x \equiv 1 \bmod 3 \cdot 2^L$).

Theoretical cost of a full pairing. The exact cost depends on the particular choice of the seed x. In any case, it boils down to the cost of $\mathbb{F}_{p^k}$ and $\mathbb{G}_2$ arithmetic operations. We follow the estimate in [27] for these operations. We model the cost of arithmetic in a degree 12, resp. degree 24 extension in the usual way, where multiplications and squarings in quadratic and cubic extensions are obtained

recursively with Karatsuba and Chung–Hasan formulas, summarized in Table 4. We denote by $\mathbf{m}_k$, $\mathbf{s}_k$, $\mathbf{i}_k$ and $\mathbf{f}_k$ the costs of multiplication, squaring, inversion, and p-th power Frobenius in an extension $\mathbb{F}_{p^k}$, and by $\mathbf{m} = \mathbf{m}_1$ the multiplication in a base field $\mathbb{F}_p$. We neglect additions and multiplications by small constants. This estimation does not include the new interleaved multiplication in extension fields by Longa [34].

Table 4. Cost from [27, 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 5. Miller loop cost in non-affine, Weierstrass model [16,4]. For $6 \mid k$, two sparse-dense multiplications cost $26\mathbf{m}_{k/6}$ whereas one sparse-sparse and one multiplication cost $6\mathbf{m}_{k/6} + \mathbf{m}_k = 24\mathbf{m}_{k/6}$.

k	$-D$	curve	DoubleLine and AddLine	ref	SparseM and SparseSparseM
$6 \mid k$	-3	$Y^2 = X^3 + b'$	$3\mathbf{m}_{k/6} + 6\mathbf{s}_{k/6} + (k/3)\mathbf{m}$ $11\mathbf{m}_{k/6} + 2\mathbf{s}_{k/6} + (k/3)\mathbf{m}$	[4, §4]	$13\mathbf{m}_{k/6}$ $6\mathbf{m}_{k/6}$

5 Algebraic tori and pairings

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 the literature [39] and [38].

Definition 2. *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 1 ([38, Lemma 1]). *Let* $\alpha \in \mathbb{F}_{p^k}^*$*, then* $\alpha^{(p^k-1)/\Phi_k(p)} \in T_k(\mathbb{F}_p)$ *.*

Lemma 2 ([38, Lemma 2]). $d \mid k \implies T_k(\mathbb{F}_p) \subseteq T_{k/d}(\mathbb{F}_{p^d})$ *.*

Corollary 1. *After the easy part of the final exponentiation in the pairing computation (Eq. 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* $d \mid k$*,* $d \neq k$*.*

5.1 Torus-based arithmetic

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 1. Rubin and Silverberg introduced in [39] a torus-based cryptosystem, 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

$$\begin{aligned} \texttt{Inverse} \quad & g \mapsto -g; \\ \texttt{Square} \quad & g \mapsto \tfrac{1}{2}(g + \gamma/g); \\ \texttt{Frobenius map} \quad & g \mapsto \frac{g^{p^i}}{\gamma^{(p^i-1)/2}} \ . \end{aligned}$$

6 Pairings in R1CS

In Section 4, we presented results from the literature that yield the most efficient pairing computation on inner BLS 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. Next, 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 $\texttt{C}$ the number of multiplication gates and take the example of a BLS12 curve.

6.1 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 $2\texttt{C}$. First $1\texttt{C}$ for the multiplication $x \cdot y$ where y is provided as an input and then $1\texttt{C}$ 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 $3\texttt{C}$ while the later costs $2\texttt{C}$ 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 $3\texttt{C}$, a squaring $2\texttt{C}$, an inversion and a division $5\texttt{C}$ ($2\texttt{C}$ 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 6 summarizes the cost of $\mathbb{G}_2$ arithmetic in R1CS. Note that a doubling is more costly in R1CS

Table 6. $\mathbb{G}_2$ arithmetic cost in R1CS over $\mathbb{F}_{p^2}$

	Div (5C)	Square (2C)	Mul (3C)	total
Double	1	2	1	12C
Add	1	1	1	10C

than an addition because the tangent slope λ requires a squaring and a division instead of just a division. The Miller function 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 $[2]S + Q$ (22C) is computed but instead we can compute $(S + Q) + S$ which costs 20C. Moreover, we can omit the computation of the y-coordinate of $S + Q$ as pointed out in a different context in [17].

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 [1], 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 [1] 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.

Table 7. $\mathbb{F}_{p^{12}}$ arithmetic cost in R1CS

	Mul	Square	Div	sparse Mul
$\mathbb{F}_{p^{12}}$	54C	36C	66C	30C

6.2 Final exponentiation

Easy part. The easy part (Eq. 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:

$$\begin{aligned} t &\leftarrow \bar{m} && (0\mathtt{C}) \\ m &\leftarrow 1/m && (66\mathtt{C}) \\ t &\leftarrow t \cdot m && (54\mathtt{C}) \\ m &\leftarrow t^{p^2} && (0\mathtt{C}) \\ m &\leftarrow t \cdot m && (54\mathtt{C}) \end{aligned}$$

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. 2. Only the multiplications and cyclotomic squarings increase the number of constraints. Squarings in cyclotomic subgroups are well studied in the literature and in Table 8 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 [21] is preferred while compression-based methods are better for other cases. For 3 squares in a row the SQR12345 variant of the Karabina method [30] 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 8. Squaring costs in the cyclotomic subgroup of $\mathbb{F}_{p^{12}}$ in R1CS

	Compress	Square	Decompress
Karatsuba + Chung–Hasan	0	36C	0
Granger-Scott [21]	0	18C	0
Karabina [30] (SQR2345)	0	12C	19C
Karabina [30] (SQR12345)	0	15C	8C

$\mathbb{T}_2$ *arithmetic.* Corollary 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 9. The $\mathbb{T}_2$ formulas are well defined over

Table 9. $\mathbb{T}_2$ arithmetic cost in R1CS.

	Compress	Square	Mul	Decompress
$\mathbb{T}_2$	24C	24C	42C	48C

$G_{q,2} \setminus \{-1, 1\}$ 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 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 [38]. 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

$$\begin{aligned} m^{p^6-1} &= (m_0 + wm_1)^{p^6-1} \\ &= (m_0 + wm_1)^{p^6}/(m_0 + wm_1) \\ &= (m_0 - wm_1)/(m_0 + wm_1) \\ &= (-m_0/m_1 + w)/(-m_0/m_1 - w) \end{aligned}$$

Hence we can absorb the $\mathbb{T}_2$ compression cost when carrying the easy part computation

$$\begin{aligned} \zeta(m^{(p^6-1)(p^2+1)}) &= (-m_0/m_1)^{p^2+1} \\ &= (-m_0/m_1)^{p^2} \cdot (-m_0/m_1) \end{aligned}$$

This costs only 60C in comparison of 120C previously. In [38], 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 show how to perform a 2-input (bits) 1-output conditional statement in R1CS in Alg. 4. This is a constant 2-bit lookup

Algorithm 4: `Lookup2`: 2-bit lookup table in R1CS

Input: bits (b_0, b_1), and constants (c_0, c_1, c_2, c_3)

Output: $r = \begin{cases} c_0, & \text{if } b_0 = 0, b_1 = 0 \\ c_1, & \text{if } b_0 = 1, b_1 = 0 \\ c_2, & \text{if } b_0 = 0, b_1 = 1 \\ c_3, & \text{if } b_0 = 1, b_1 = 1 \end{cases}$

1 $t_1, t_2 \leftarrow$ temporary variables;
2 $(c_3 - c_2 - c_1 + c_0) \times b_1 = t_1 - c_1 + c_0$;
3 $t_1 \times b_0 = t_2$;
4 $(c_2 - c_0) \times b_1 = r - t_2 - c_0$;
5 **return** r;

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 Implementation and benchmark

To the best of our knowledge, there are only two implementations of pairings in R1CS. One in `libsnark` [7] for Miyaji–Nakabayashi–Takano [37] curves of embedding degrees 4 (MNT4) and 6 (MNT6) and one for BLS12-377 in `arkworks` [15]. The first one was written in C++ 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. More discussion on this can be found in this survey paper [3, Section 5]. The second implementation is in Rust and corresponds exactly to the problem we investigate in this paper. 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 the open-source `gnark` ecosystem [10]. 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 [24] verification, BLS signature verification and KZG polynomial commitment opening. Tables 10 and 11 give the overall cost of these circuits in terms of number of constraints `C`, which is almost half the best previously known implementation cost. The source code was merged into the `gnark` standard library:

`https://github.com/ConsenSys/gnark/tree/master/std`

Table 10. Pairing cost for BLS12-377 and BLS24-315 in R1CS.

	Miller loop	Final exponentiation	total
`arkworks` (BLS12-377)	$\approx$ 6000`C`	$\approx$ 13000`C`	$\approx$ 19000`C`
`gnark` (BLS12-377)	5519`C`	6016`C`	11535`C`
`gnark` (BLS24-315)	8132`C`	19428`C`	27608`C`

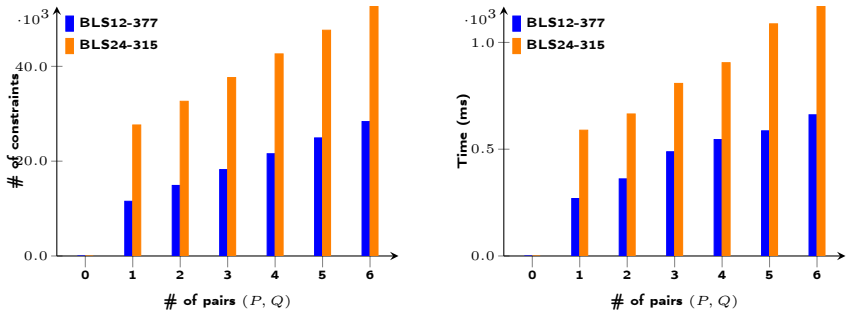
Table 11. Pairing-based circuits costs in R1CS for BLS12-377 and BLS24-315.

	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 [12, Sec. 6.2 - Alg. 1].

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 Fig. 4 the 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 the open-source library `gnark` [10] where we implemented the pairings 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.

Fig. 4. Groth16 number of constraints (a) and proving times (b) for multi-pairings on BLS12-377 and BLS24-315 curves.

8 Conclusion

The application of pairing-based cryptography in modern zero-knowledge proof systems has energized research efforts well beyond traditional use of pairings. In recursive pairing-based proof systems one requires to efficiently prove a pairing computation. To this end researchers have come up with new tailored constructions of elliptic curves to allow proving a pairing efficiently in a generic-purpose proof system. However, once these curves are constructed, so little work was conducted in order to optimize the pairing computation in the *arithmetization* model of the proof system. In this work we considered the Rank-1 constraint system as a widely used arithmetization model and reduced the computational cost beyond the state-of-the-art. As a future work, we are looking to consider other models such as the PLONK [19] arithmetization where additions are not free but where we can build custom gates and lookup tables for some specific intermediate computations.

References

1. Aranha, D.F., Barreto, P.S.L.M., Longa, P., Ricardini, J.E.: The realm of the pairings. In: Lange, T., Lauter, K., Lisonek, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 3–25. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-43414-7_1

2. Aranha, D.F., Fuentes-Castañeda, L., Knapp, E., Menezes, A., Rodríguez-Henríquez, F.: Implementing pairings at the 192-bit security level. In: Abdalla, M., Lange, T. (eds.) PAIRING 2012. LNCS, vol. 7708, pp. 177–195. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-36334-4_11
3. Aranha, D.F., Housni, Y.E., Guillevic, A.: A survey of elliptic curves for proof systems. Cryptology ePrint Archive, Paper 2022/586 (2022), https://eprint.iacr.org/2022/586, https://eprint.iacr.org/2022/586
4. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López-Hernández, J.C.: Faster explicit formulas for computing pairings over ordinary curves. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 48–68. Springer, Heidelberg (May 2011). https://doi.org/10.1007/978-3-642-20465-4_5
5. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–368. Springer, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_23
6. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 02. LNCS, vol. 2576, pp. 257–267. Springer, Heidelberg (Sep 2003). https://doi.org/10.1007/3-540-36413-7_19
7. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M., Wu, H., Contributors: C++ library for zksnark, https://github.com/scipr-lab/libsnark
8. Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (Feb 2005). https://doi.org/10.1007/978-3-540-30576-7_18
9. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_30
10. Botrel, G., Piellard, T., Housni, Y.E., Kubjas, I., Tabaie, A.: Consensys/gnark (Feb 2022). https://doi.org/10.5281/zenodo.6093969, https://doi.org/10.5281/zenodo.6093969
11. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: ZEXE: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy. pp. 947–964. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00050
12. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021 (2019), https://eprint.iacr.org/2019/1021
13. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45721-1_26
14. Chung, J., Hasan, M.A.: Asymmetric squaring formulae. In: 18th IEEE Symposium on Computer Arithmetic (ARITH '07). pp. 113–122 (2007). https://doi.org/10.1109/ARITH.2007.11
15. arkworks Contributors: arkworks zkSNARK ecosystem. https://arkworks.rs (2022)
16. Costello, C., Lange, T., Naehrig, M.: Faster pairing computations on curves with high-degree twists. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 224–242. Springer, Heidelberg (May 2010). https://doi.org/10.1007/978-3-642-13013-7_14

17. Eisenträger, K., Lauter, K., Montgomery, P.L.: Fast elliptic curve arithmetic and improved Weil pairing evaluation. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 343–354. Springer, Heidelberg (Apr 2003). https://doi.org/10.1007/3-540-36563-X_24
18. El Housni, Y., Guillevic, A.: Families of SNARK-friendly 2-chains of elliptic curves. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022. LNCS, vol. 13276, pp. 367–396. Springer (2022). https://doi.org/10.1007/978-3-031-07085-3_13, ePrint 2021/1359
19. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. ePrint 2019/953
20. Ghammam, L., Fouotsa, E.: 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
21. Granger, R., Scott, M.: Faster squaring in the cyclotomic subgroup of sixth degree extensions. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 209–223. Springer, Heidelberg (May 2010). https://doi.org/10.1007/978-3-642-13013-7_13
22. Grewal, G., Azarderakhsh, R., Longa, P., Hu, S., Jao, D.: Efficient implementation of bilinear pairings on ARM processors. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 149–165. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-35999-6_11
23. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_19
24. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_11
25. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 698–728. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_24
26. Groth, J., Ostrovsky, R., Sahai, A.: Non-interactive zaps and new techniques for NIZK. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 97–111. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_6
27. Guillevic, A., Masson, S., Thomé, E.: Cocks–Pinch curves of embedding degrees five to eight and optimal ate pairing computation. Des. Codes Cryptogr. **88**, 1047–1081 (March 2020). https://doi.org/10.1007/s10623-020-00727-w
28. Hayashida, D., Hayasaka, K., Teruya, T.: Efficient final exponentiation via cyclotomic structure for pairings over families of elliptic curves. ePrint 2020/875
29. Housni, Y.E., Guillevic, A.: Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 20. LNCS, vol. 12579, pp. 259–279. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-65411-5_13
30. Karabina, K.: Squaring in cyclotomic subgroups. Math. Comput. **82**(281), 555–579 (2013). https://doi.org/10.1090/S0025-5718-2012-02625-1
31. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_11

32. Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., shelat, a., Shi, E.: C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093 (2015), https://eprint.iacr.org/2015/1093
33. Liochon, N., Chapuis-Chkaiban, T., Belling, A., Begassat, O.: A zk-evm specification. https://ethresear.ch/t/a-zk-evm-specification/11549 (2021)
34. Longa, P.: Efficient algorithms for large prime characteristic fields and their application to bilinear pairings and supersingular isogeny-based protocols. Cryptology ePrint Archive, Report 2022/367 (2022), https://ia.cr/2022/367
35. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3339817
36. Miller, V.S.: The Weil pairing, and its efficient calculation. Journal of Cryptology **17**(4), 235–261 (Sep 2004). https://doi.org/10.1007/s00145-004-0315-8
37. Miyaji, A., Nakabayashi, M., Takano, S.: Characterization of elliptic curve traces under FR-reduction. In: Won, D. (ed.) ICISC 00. LNCS, vol. 2015, pp. 90–108. Springer, Heidelberg (Dec 2001)
38. Naehrig, M., Barreto, P.S.L.M., Schwabe, P.: On compressible pairings and their computation. In: Vaudenay, S. (ed.) AFRICACRYPT 08. LNCS, vol. 5023, pp. 371–388. Springer, Heidelberg (Jun 2008)
39. Rubin, K., Silverberg, A.: Torus-based cryptography. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 349–365. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_21
40. Scott, M.: Pairing implementation revisited. ePrint 2019/077
41. Scott, M.: A note on twists for pairing friendly curves (2009), http://indigo.ie/~mscott/twists.pdf
42. Stam, M.: XTR and tori. Cryptology ePrint Archive, Report 2021/1659 (2021), https://eprint.iacr.org/2021/1659
43. Vercauteren, F.: Optimal pairings. IEEE Transactions on Information Theory **56**(1), 455–461 (Jan 2010). https://doi.org/10.1109/TIT.2009.2034881
44. Xiong, A.L., Chen, B., Zhang, Z., Bünz, B., Fisch, B., Krell, F., Camacho, P.: VERI-ZEXE: Decentralized private computation with universal setup. Cryptology ePrint Archive, Report 2022/802 (2022), https://eprint.iacr.org/2022/802