On Proving Pairings

Andrija Novakovic*[1] and Liam Eagen†[2]

[1]Geometry Research
[2]Alpen Labs, Zeta Function Technologies

April 24, 2024

Abstract

In this paper we explore efficient ways to prove correctness of elliptic curve pairing relations. Pairing-based cryptographic protocols such as the Groth16 and Plonk SNARKs and the BLS signature scheme are used extensively in public blockchains such as Ethereum due in large part to their small size. However the relatively high cost of pairing computation remains a practical problem for many use cases such as verification "in circuit" inside a SNARK. This naturally arises in recursive SNARK composition and SNARKs of BLS based consensus protocols.

To improve pairing verification, we first show that the final exponentiation step of pairing verification can be replaced with a more efficient "residue check," which can be incorporated into the Miller loop. Then, we show how to reduce the cost of the Miller loop by pre-computing all the necessary lines, and how this is especially efficient when the second pairing argument is fixed in advance. This is the case for BLS signatures with a fixed public key, as well as for KZG based SNARKs like Plonk and two of the three Groth16 pairings. Finally, we show how to improve of the protocol of [gar] by combining quotients, which allows us to more efficiently prove higher degree relations. These techniques also carry over naturally to pairing verification, for example on-chain verification or as part of the BitVM(2) protocol for Bitcoin smart contracts. We instantiate algorithms and show results for the BN254 curve.

*andrija@geometry.dev
†liameagen@protonmail.com

Contents

1 Introduction

Pairing based cryptography has found numerous applications, in particular with respect to public blockchains. The two primary uses of pairing based cryptography in blockchains are the BLS[BLS04] signature scheme and Succinct Non-interactive Arguments of Knowledge (SNARKs) like Groth16 [Gro16] and Plonk [GWC19]. In both cases, the additional structure that pairings provide over ordinary elliptic curves allows for lower communication costs. For example, BLS signatures support non-interactive aggregation which is exploited by Ethereum's Proof-of-Stake (PoS) protocol, and pairing based SNARKs yield protocols with constant sized, concretely small proofs and with constant time verifiers. Often, when other proof systems are used, they are wrapped with a pairing based SNARK before being posted on chain. However, these benefits come at the cost of a much more computationally expensive and mathematically complicated primitive.

More recently, pairings have been used both to instantiate proof systems and as the target of proof systems. For example, if one wanted to construct a proof of Ethereum's consensus protocol to construct a bridge to another blockchain [Suc], they would need to prove the correct execution of a pairing. One might also need to produce such a proof when constructing a recursive zero knowledge proof. That is, a proof that proves knowledge of some number of distinct, valid proofs. In some cases it is possible to avoid proving pairings in circuit, for example by using folding schemes [KST21], but often it is not. When aggregating different types of proofs [BCL$^+$20] in different proof systems it is often impossible to avoid verifying pairings in circuit.

Pairing computations, except for the Weil pairing [Mil04], consists of two parts: the Miller loop and the final exponentiation. Of the two, the final exponentiation is generally more expensive, especially when using optimal pairing optimizations [Ver08]. Existing implementation of pairings in arithmetic circuits such as [Hou22] and [gar] adopt the straightforward approach of directly checking a pairing computation in a circuit. That is, they first check that each step of the Miller loop was correctly computed, and then check that the final exponentiation was correctly computed. Our approach is able to replace the final exponentiation computation with one short exponentiation and substantially reduce the cost of verifying the Miller loop.

1.1 Contributions

Our primary contribution is a more computationally efficient protocol for verifying elliptic curve pairings. To construct this protocol, we exploit the highly structured nature of pairing computation, and exploit the fundamental differences between *computing* pairings and *verifying* pairings. This protocol can be used to reduce the arithmetic circuit size required to prove pairings within SNARKs, and can be used to reduce the costs of verifying pairings in extremely resource constrained environments. In particular, our techniques can help to reduce the cost of on-chain verification of pairings, both directly as in Ethereum and via higher level protocols like BitVM or BitVM2 [bita] on Bitcoin. In so doing, we introduce a number of optimizations that may be of independent interest. Throughout, we focus on the BN family of curves, but all our optimizations readily generalize.

Final Exponentiation First, we replace the final exponentiation of the Tate pairing with an r-residue check. This immediately replaces the final exponentiation with an exponentiation one third the length in the case of BN curves. Then, we show how to use the same Frobenius structure of an optimal pairing to perform the r residue check even more efficiently, further reducing the exponent by a factor 4 for BN. Finally, we show how the r-residue check can integrated into the Miller loop directly, saving all the extension field squaring operations. In cases where the

optimal ate lattice has very low hamming weight, this almost entirely eliminates the cost of the final exponentiation.

Line Precomputation Second, we replace the Miller loop's line computation with "witnessing" lines. That is, providing them as auxiliary inputs to the circuit. In the case of variable second argument pairings this produces a modest savings since verifying the correctness of the lines is cheaper than computing them. In the case of fixed second argument pairings, as in most KZG SNARKs and significant part of Groth16, this allows us to pre-compute all the lines and eliminate much of the overall cost of the Miller loop.

Randomized Extension Arithmetic Third, we improve on the work of Garaga [gar] to reduce the cost of extension field arithmetic for pairings. Similarly, we exploit the isomorphism between elements of extension fields and elements of polynomial quotient rings, which allows us to test extension field relations by evaluating polynomials at a random challenge. However, we use an additional challenge to first combine all the relations into a single random linear combination and provide a single quotient. This allows us reduce commitment costs by both avoiding committing to $O(n)$ quotients and by using higher degree $\mathbb{F}_{q^k}$ arithmetic.

1.2 Related Work

Housni [Hou22] analyzed the costs of computing pairings over Rank-1 Constraint Systems. Their analysis has the same observation that affine coordinate representation of points yields a better performance in circuits given that inversions can be replaced with multiplications. They further build efficient short addition chains that optimize number of additions instead of doublings, and show that computation of $(S+Q)+S$ yields a slightly better result than computation of $2S+Q$. Circom-pairing [Cir] also uses affine coordinate representation and they use an approach similar to ours of showing that points are co-linear.

Garaga [gar] utilizes randomized polynomial testing technique which reduces $\mathbb{F}_{q^k}$ arithmetic to testing equality of polynomials in $\mathbb{F}_q$ by the Schwartz-Zippel lemma. We note that they commit to a quotient polynomial per multiplication whereas we commit to single quotient polynomial by utilizing one separation and one evaluation challenge. All of the above mentioned approaches perform the final exponentiation inside the circuit.

2 Techniques

Tate pairing computation is expensive, but it is also highly structured, and we are able to exploit this structure to efficiently verify pairings. Most of our contributions follow a similar pattern: rather than check some expensive computation directly by carrying it out, we provide some auxiliary information and use that to verify the computation much more efficiently. This form of optimization is very common in SNARK design, for example in how most SNARKs check modular inversion. Computing an inverse is, relatively to multiplication, a very expensive operation and involves either computing the extended euclidean algorithm or a large exponentiation. By contrast, checking an inversion is very simple: to show that $y^{-1} = x$ we can simply check $xy = 1$.

2.1 Eliminating the Final Exponentiation

The first optimization we introduce of this form allows us to eliminate the so called 'final exponentiation' of pairing computation. The Tate pairing, and related pairings like the Ate pairing,

output an element of an equivalence class $\mathbb{F}^*/(\mathbb{F}^*)^r$. Two elements $x, y \in \mathbb{F}$ are equivalent if there exists some $c \in \mathbb{F}^*$ such that $xc^r = y$. Because the Tate pairing is only well defined up to this equivalence, we need some way to normalize its output to check for equality. This normalization is called the final exponentiation, and consists of clearing the r residue part of the output by computing $x \mapsto x^{(q^k-1)/r}$. In our previous example we see that if $xc^r = y$ then after the final exponentiation

$$y^{(q^k-1)/r} = (xc^r)^{(q^k-1)/r} = x^{(q^k-1)/r} c^{q^k-1} = x^{(q^k-1)/r}.$$

The final exponentiation is very expensive, both because $q^k - 1$ is concretely very large and all arithmetic is carried out over $\mathbb{F}_{q^k}$. While there are some tricks that one can exploit to reduce this cost by using cyclotomic factors of $q^k - 1$, it remains large. Our optimization avoids this cost by instead providing c as auxiliary input and directly checking $xc^r = y$. In this way we replace an exponentiation by $(q^k - 1)/r$ with an exponentiation by r, which in general is much cheaper.

Note that this optimization admits a convenient generalization: to show x, y are equivalent we can instead show $xc'^s = y$ where $s = rt$. This follows straightforwardly since letting $c = c'^t$ gives $xc^r = y$. In many cases over an extension field, there exist s of this form where computing x^s is much cheaper than computing x^r. This family of optimizations is essentially the same as that used to construct the Ate pairing [HSV06] and involves finding a short linear combination of Frobenius endomorphisms that vanishes $\bmod r$. This poses some technical issues if $\gcd(t, q^k - 1) \neq 1$, but nevertheless yields substantially better performance than raising to r. We use the same linear combination to generate s and to instantiate the Ate pairing, which yields further performance improvements.

2.2 Precomputing Lines

The other part of pairing computation is the Miller loop. When computing a pairing $e(P, Q)$, at each step of the Miller loop computation we need to compute a line through some points T, Q and then to evaluate the line at P. Instead of computing the lines, we simply provide the coefficients λ, μ as auxiliary input. Then, we show that the line $l : y = \lambda \cdot x + \mu$ passes through T and Q by checking $l(T) = \mathcal{O}$ and $l(Q) = \mathcal{O}$. From this, we can easily evaluate $l(P)$. We further show that the optimizations of various works [ABLR13, BDM+10] that exploit twists and line normalizations are also compatible with this technique in section 5.

We extend this optimization to the situation when the second argument of the pairing is fixed. This is the case in SNARKs based on KZG [KZG10] such as PLONK [GWC19]. The lines used by the Miller loop computation only depend on the second argument to the pairing, so when it is fixed we can precompute all the lines and drop the $l(T) = l(Q) = \mathcal{O}$ checks. It is also possible to "hard code" the lines into the computation, which is especially useful in Bitcoin script where input size is severely constrained.

3 Preliminaries

In this section we briefly revisit some definitions and facts about the pairings that will be useful later. We will assume some familiarity with number theory and elliptic curves, and refer the reader to Washington [Was03] for more background on this topic.

3.1 Elliptic Curves

Let $q = p^n$ and $\mathbb{F}_q$ be a finite field of characteristic $p > 3$ with q elements and $\gcd(n, p) = 1$. Let E be an elliptic curve defined over $\mathbb{F}_q$ and $E(\mathbb{F}_q)$ the group of points over this field. Suppose

that $r \mid \#E(\mathbb{F}_q)$ and $r^2 \nmid \#E(\mathbb{F}_q)$. It therefore follows that the full r-torsion $E[r]$ of the curve is defined over $\mathbb{F}_{q^k}$, i.e. $E[r] \subset E(\mathbb{F}_{q^k})$

Recall that a divisor of an elliptic curve is a formal integer linear combination of points on the curve, and that a divisor is principal if it is precisely the zeros (poles) of a rational function of the curve. It is well known that the group of elliptic curve points is isomorphic to the divisor class group of the curve. This follows with some work from the familiar formula for Weierstrass point addition, since each line is a rational function of the curve and $P, Q, -(P+Q)$ are collinear.

3.2 Tate Pairing

For every $P \in E(\mathbb{F}_{q^k})$, define $f_{s,P}$ to be a rational function with divisor

$$(f_{s,P}) = s(P) - ([s]P) - [s-1](\mathcal{O})$$

This function is called Miller function and it is determined uniquely up to multiplication by non zero elements of $\mathbb{F}_{q^k}$. Note that this function is well defined for all s and P since it has degree 0, $s - 1 - (s-1) = 0$, and $s(P) - (sP) = \mathcal{O}$. Additionally, when P is of order s the divisor simplifies to $(f_{s,P}) = s(P) - s(\mathcal{O})$.

Evaluation of $f_{s,P}$ is well defined for all $Q \neq P, \mathcal{O}$. To define the Tate pairing of P with Q, let $R \neq \mathcal{O}, P - Q$. We then define the pairing to be

$$t(P, Q) = f_{r,P}(Q + R)/f_{r,P}(R).$$

This is a well defined, non-degenerate bilinear pairing of the following groups:

$$t : E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \to \mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r.$$

Note that the output of the pairing is an element of the quotient group $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$. This means that outputs of the pairing are only unique up to multiplication by an r residue. To test whether two pairings produce the same output, it is necessary to normalize the output by performing a final exponentiation. Raising to the $h = (q^k - 1)/r$ clears the r-residue part of the output of the pairing leaving only an r-root of unity.

When R lies in a proper subfield of $\mathbb{F}_{q^k}$ and $f_{s,P}$ is suitably normalized, then $f_{s,P}(R)$ also lies in the subfield. In that case, $f_{s,P}(R)^{(q^k-1)/r} = 1$ for all R and we can simply define the pairing as

$$t(P, Q) = f_{r,P}(Q).$$

3.3 The Miller Loop

In both the Tate and Ate pairings, we need to compute the functions $f_{s,Q}(P)$. While $f_{s,Q}$ is technically in the function field of the curve, it has a number of coefficients proportional to s, which is far too large to work with. To deal with this problem, Victor Miller proposed an iterative algorithm [Mil02] for computing these evaluations. He uses the additive identity of Miller functions to compute $f_{s,Q}(P)$ from smaller Miller functions and line evaluations in an addition chain. This is essentially the same principle used to compute modular exponentiations. This algorithm runs in $O(\log s)$ field multiplications and it is used for computing both Weil and Tate pairings.

By $l_{T,P}(Q)$ denote evaluation of a line passing through points T, P at Q and similarly by $v_P(Q)$ denote evaluation of a vertical line passing through P evaluated at Q. Then Miller's algorithm can be computed with Algorithm 1:

Algorithm 1: Miller's algorithm for elliptic curves

Input: $r = \sum_{i=0}^{L} s_i 2^i$ for $s_i \in \{0, 1\}$ where $s_L = 1$ and $P, Q \in E[r], P \neq Q$
Output: $f_{r,P}(Q)$

$T \leftarrow P, f \leftarrow 1$
for $i = L - 1$ **to** 0 **do**
  $f \leftarrow f^2 \cdot l_{T,T}(Q)/v_{[2]T}(Q)$
  $T \leftarrow [2]T$
  if $s_i = 1$ **then**
    $f \leftarrow f \cdot l_{T,P}(Q)/v_{T+P}(Q)$
    $T \leftarrow T + P$
  end
end
return f

Later works, like Optimal Pairings [Ver08], found more optimizations to reduce complexity of a Miller loop by shortening it's length. Barreto *et al.* [BKLS02] also observed that, when embedding degree k is even, vertical lines can be omitted since their evaluations lie in the subfield of $\mathbb{F}_{q^k}$, and therefore they vanish after the final exponentiation. We instantiate results for BN [bn] curve thus we also state the steps of optimal Miller's loop computation over BN curve. Let t be a parameter of the curve, let $\pi_p : E \mapsto E$ be the Frobenius endomorphism given by $\pi_p(x, y) \mapsto (x^p, y^p)$ and let $\lambda = 6t + 2 + p - p^2 + p^3$ [Ver08], then optimal Miller's algorithm over BN curve is obtained by evaluating $f_{\lambda,Q}(P)$ and it is can be done with the Algorithm 2:

3.4 Final Exponentiation

Computing final exponentiation naïvely would be extremely expensive, as h has about 11 times as many bits as q for BN curves. Scott *et al.* [SBC+08] are able to perform the final exponentiation much more efficiently by exploiting the cyclotomic factorization of the final exponentiation. That is, in embedding degree 12 curves they more efficiently perform $h = (q^{12} - 1)/r$ exponentiations by splitting h into three coefficients as

$$h = \frac{q^{12} - 1}{r} = (q^6 - 1) \cdot (q^2 + 1) \cdot \frac{q^4 - q^2 + 1}{r}$$

This yields a number of convenient properties and opportunities for optimization. For example, after raising a field element to the power $(q^6 - 1)$ it becomes a 'unitary element' where inverses can be computed via conjugation. As we discussed in the technical overview, computing frobenius operations is essentially free. Since they are able to perform inversions very cheaply, they can use signed digits when computing the exponentiation.

While this optimization is crucially important for *computing* pairings, it is not necessary for *verifying* pairings. This is because inversions cost no more than a single multiplication to check. Nevertheless, we also use signed digits when computing the final exponentiation, and later the Miller loop. The steps are outlined in the Algorithm 3.

3.4.1 Frobenius operator

The efficiency of pairing computation lies in the fact that computing c^q for arbitrary c in the quadratic extension field $\mathbb{F}_{q^2}$, which is defined by a quadratic polynomial of the form $x^2 - u = 0$,

Algorithm 2: Optimal ate Miller loop over Barreto–Naehrig curves.

Input: $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$
Output: $f_{\lambda,Q}(P)$

Write $s = 6t + 2 = \sum_{i=0}^{L} s_i 2^i$ **where** $s_i \in \{-1, 0, 1\}$ **and** $s_L = 1$
$T \leftarrow Q, f \leftarrow 1$
for $i = L - 2$ **to** 0 **do**
    $f \leftarrow f^2 \cdot l_{T,T}(P)$
    $T \leftarrow [2]T$
    if $s_i = 1$ **then**
        $f \leftarrow f \cdot l_{T,Q}(P)$
        $T \leftarrow T + Q$
    end
    if $s_i = -1$ **then**
        $f \leftarrow f \cdot l_{T,-Q}(P)$
        $T \leftarrow T - Q$
    end
end
$Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow \pi_p(Q_1), Q_3 \leftarrow \pi_p(Q_2)$
$f \leftarrow f \cdot l_{T,Q_1}(P), T \leftarrow T + Q_1$
$f \leftarrow f \cdot l_{T,-Q_2}(P), T \leftarrow T - Q_2$
$f \leftarrow f \cdot l_{T,Q_3}(P), T \leftarrow T + Q_3$
return f

Algorithm 3: Exponentiation in $\mathbb{F}_{q^k}$

Input: $a \in \mathbb{F}_{q^k}, a^{-1} \in \mathbb{F}_{q^k}$, $e = \sum_{i=0}^{L-1} s_i 2^i$ for $s_i \in \{-1, 0, 1\}$
Output: a^e

$assert!(a \cdot a^{-1} = 1)$
$c = a$
for $i = L - 2$ **to** 0 **do**
    $c \leftarrow c^2$
    if $s_i = -1$ **then**
        $c \leftarrow c \cdot a^{-1}$
    end
    else if $s_i = 1$ **then**
        $c \leftarrow c \cdot a$
    end
end
return c

is simply conjugation [GG87, Theorem 11.6]. More generally for $c = c_0 + uc_1$ it follows that $(c)^{q^{2i}} = c$ and $(c)^{q^{2i-1}} = \overline{c}$ for $\overline{c} = c_0 - uc_1$ and $i \in \mathbb{N}$. Beuchat *et al.* [BDM+10] further use the fact that extension $\mathbb{F}_{q^k}$ can be represented as a sextic extension of the quadratic field, i.e., $\mathbb{F}_{q^k} = \mathbb{F}_{q^2}[W]/(W^6 - u)$. Then an element from $\mathbb{F}_{q^k}$ can be represented as $\sum_{i=0}^{5} a_i W^i$ for $W^p = u^{(p-1)/6}W$ and $a_i \in \mathbb{F}_{q^2}$. After precomputation of $\gamma_i = u^{i(p-1)/6}$ for $i \in [1, 5]$, the Frobenius operator can be computed as follows:

$$f^p = (a_0 + a_1 W + a_2 W^2 + a_3 W^3 + a_4 W^4 + a_5 W^5)^5 = \sum_{i=0}^{5} \gamma_i \overline{a_i} W^i$$

Which requires just a few $\mathbb{F}_q$ multiplications and $\mathbb{F}_{q^2}$ conjugations. Similar asymptotics are obtained for other powers of q^i and we use identical approach for raising to power q^i in our paper. Exact steps can be found in the paper [BDM+10, Appendix A].

3.5 Ate Pairing

The Ate pairing is simply an optimized version of Tate pairing where we replace r with some multiple of r that is more efficient to compute. This works because of two identities of Miller functions:

$$f_{a+b,Q}(P) = f_{a,Q}(P) f_{b,Q}(P) l_{aQ,bQ}(P)$$
$$f_{ab,Q}(P) = f_{a,Q}(P)^b f_{b,aQ}(P).$$

In particular when $rQ = \mathcal{O}$, it follows from the second that $f_{rt,Q}(P) = f_{r,Q}(P)^t f_{t,rQ}(P) = f_{r,Q}(P)^t$. The trick of the Ate pairing is finding some $s = rt$ such that $f_{s,Q}$ is more efficient to compute by exploiting the Frobenius endomorphism

Specifically, we want to find some short linear combination of powers of $q \bmod r$ that vanishes

$$\sum_{i=0}^{\varphi(k)-1} c_i q^i \equiv 0 \mod r.$$

Then by applying the identities of the Miller function, we can write $f_{s,Q}$ in terms of $f_{c_i,Q}^{q^j}$ which are all much cheaper to evaluate. Concretely, many pairing curves used in practice have very low hamming weight c_i that dramatically improve performance of pairings. So long as $\gcd(t, r) = 1$, this transformation of the problem is sound.

The specifics of this are outside the scope of this paper and we point to Optimal Pairings [Ver08] for more details.

4 Eliminating the Final Exponentiation

To test whether two pairings are equal, we need to test if their outputs lie in the same equivalence class. Conventionally one does this by normalizing them via a final exponentiation by $h = (q^k - 1)/r$ since $a_0 \sim a_1 \implies a_0^h = a_1^h$ as

$$(a_0^h / a_1^h) = b^{rh} = b^{q^k - 1} = 1.$$

In this section, we show how to replace this final exponentiation with a series of progressively less expensive, equivalent checks. We have tried to keep this section relatively self contained

with respect to number theoretic background information. However, for interested readers we recommend [HH39] for additional background on number theory and [Was03] for background on elliptic curves.

The Miller loop for the Tate pairing, and its optimal variants, outputs an element of $\mathbb{F}_{q^k}$ not necessarily an r root of unity. As a result, equivalent Miller loop computations may output different elements of $\mathbb{F}_{q^k}$. However, the non-degeneracy of the pairing guarantees that the r root of unity part of these outputs will be the same. Equivalently, we can say that the output of the Miller loop is a member of an equivalence class in the group $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$. Here $(\mathbb{F}_{q^k}^*)^r = \{c^r : c \in \mathbb{F}_{q^k}^*\}$, so two elements $a_1 \sim a_2$ are equivalent if there exists some $b \in \mathbb{F}_{q^k}$ such that $a_1/a_2 = b^r$. Now we can recover the formal definition of the un-reduced Tate pairing

$$t : \mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r.$$

4.1 Soundness of Residue Check

The core insight of our paper is that in order to test if pairings are equal, it is enough to test if outputs of the Miller function lie in the same equivalence class directly. This is expressed in the following theorem, which states that a pairing product equals 1 if and only if the product of the associated Miller loops equals an r residue.

Theorem 1.

$$\prod_i e(P_i, Q_i) = 1 \iff \exists c \in \mathbb{F}_{q^k}^* : \prod_i f_{r,Q_i}(P_i) = c^r$$

Proof. Let $x = \prod_i f_{r,Q_i}(P_i)$

Completeness (if direction) Suppose $1 = \prod_i e(P_i, Q_i) = \left(\prod_i f_{r,Q_i}(P_i)\right)^h = x^h$. Since $r^2 \nmid p^k - 1$ we have that $\gcd(r, h) = 1$ and $h' = h^{-1} \mod r$ is well defined. Then observe that $x^{hh'} = 1^{h'} = 1$. Note that $hh' = 1 + rs$ by construction so $x^{hh'} = x^{1+rs}$ so $c = x^{-s}$ and $x = c^r$.

Soundness (only if direction) Suppose $\prod_i f_{r,Q_i}(P_i) = c^r$. Raising both sides to the power of h we find that $\prod_i e(P_i, Q_i) = \left(\prod_i f_{r,Q_i}(P_i)\right)^h = c^{rh} = 1$. □

Note that the effective form of the theorem can be obtained via contraposition. That is, to show that if the pairing product is not 1 then there is no c^r simply take the contraposition of the "only if" direction.

4.2 Optimal Exponentiation

In the previous section we showed that the final exponentiation can be replaced with just computing c^r, and in this section we develop a method for optimal exponentiation by r. The straightforward way of computing it with square and multiply algorithm 3 doesn't produce a significant reduction in number of $\mathbb{F}_{q^k}$ operations compared to computing full final exponentiation. The key insight is that we can apply the same method that is used for Miller loop shortening which utilizes the efficiency of computing Frobenius operator 3.4.1.

4.2.1 Hasse bound

To build an intuition of how the exponentiation c^r can be optimized, recall the Hasse theorem [Sil09, Theorem 1.1] that provides an estimate of the number of points on elliptic curves over a finite fields.

Hasse theorem. Let E be an elliptic curve over finite field $\mathbb{F}_q$. Then the order of $E(\mathbb{F}_q)$ satisfies

$$|q + 1 - \#E(\mathbb{F}_q)| \leq 2\sqrt{q}$$

and the quantity $t = q + 1 - \#E(\mathbb{F}_q)$ is called the trace of Frobenius.

Assume that we are working with curve which cofactor is 1. Then $\#E(\mathbb{F}_q) = r$ and from the Hasse theorem we can conclude that $r = q + 1 - t$. Now recall that computing c^{q^i} 3.4.1 is significantly cheaper than the classic square and multiply approach 3. Therefore computation of c^r is dominated by the exponentiation c^t. Again, from Hasse theorem we know that t is bounded by $O(\sqrt{q})$, thus straightforward exponentiation c^r will run in $O(\log_2(q))$ where the exponentiation c^t will run in $O(\log_2(\sqrt{q}))$.

So, at high level the idea for optimization is trying to express r as some polynomial $\sum_{i=0} a_i q^i$ and try to minimize coefficients a_i.

4.2.2 Optimal λ

Optimal Pairings [Ver08] shows that the most optimal pairing is obtained from finding the shortest vector v in the lattice spanned by the following rows:

$$L := \begin{bmatrix} r & 0 & 0 & \dots & 0 \\ -q & 1 & 0 & \dots & 0 \\ -q^2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \ddots & \\ -q^{\varphi(k)-1} & 0 & \dots & 0 & 1 \end{bmatrix}$$

Then they prove that $\lambda = \sum_{i=0} v_i q^i = 0 \mod r$ and that coefficients v_i are the smallest possible. Therefore, we can use the exact same λ to obtain the most optimal exponentiation. The only problem is that $\lambda \neq r$ but λ is such that $r|\lambda$. Therefore we proceed to show that the computation c^r can be soundly replaced with computation of c^λ. Further all the squarings of the exponentiation c^λ can be embedded into computation of the Miller loop and thus we are able to replace the final exponentiation with just a few $\mathbb{F}_{q^k}$ multiplications.

Further define m such that $\lambda = m \cdot r$ and $d = \gcd(m, h)$ where $h = (q^k - 1)/r$. Then we have the following claims:

Theorem 2.

$$\prod_i e(P_i, Q_i) = 1 \iff \exists c \in \mathbb{F}_{q^k}^* : \prod_i f_{r,Q_i}(P_i) = c^{\lambda/d}$$

Proof. If $\prod_{i=0}^n e(Q_i, P_i) = 1$ then from theorem 1 we know that $\prod_{i=0}^n f_{r,Q_i}(P_i) = c^r$ and it follows that $ord(c)|h$. Further by definition of d we know that $\frac{m}{d} \nmid h$ and similarly $\frac{m}{d}$ doesn't divide any divisor of h, therefore $\gcd(\frac{m}{d}, ord(c)) = 1$. This means that there exists m' such that $\frac{m}{d} \cdot m' = 1 \mod ord(c)$. Thus for c it holds that $(c^{m'})^{(m/d)} = c$ from which we conclude that output of a Miller loop can always be reduced to $c^{m'}$ which is is $\frac{\lambda}{d}$-th residue.

On the other side raising both sides to power h, it's easy to see that $\prod_i f_{r,Q_i}(P_i)^h = (c^{\frac{m}{d}})^{rh} = 1 = \prod_{i=0}^n e(Q_i, P_i)$ □

We saw that output of the Miller loop is always $\frac{\lambda}{d}$-th residue, but in general, when $ord(c)|d$, it might not be a full λ-th residue. However the major cost reduction comes from explicitly replacing final exponentiation with the exponentiation c^{λ}. Therefore we propose a method of 'scaling' a Miller loop outputs in order to make them λ residues. We still need to make sure that this 'scaling' firstly doesn't break soundness, i.e., if the output of Miller loop wasn't a r-th residue then it should also not be an r-th residue after this modification. Secondly we prove that this operation is complete or in other words if the Miller loop output is not a λ residue honest $\mathcal{P}$ should always be able to perform 'scaling' and make it a λ residue. Last but not least, 'scaling' and proving that the output is a λ residue (i.e., computing c^{λ}) should still be cheaper than proving that Miller loop outputs are r-th residues (i.e., computing c^{r}).

With the following lemma we prove that if the Miller loop output is not r-th residue, then multiplying it with r-th residues can't make it one.

Lemma 1. *Let a and u be such that $a^r = u$, If $\prod_{i=0}^{n} e(Q_i, P_i) \neq 1$ then there is no $c \in \mathbb{F}_{q^k}^*$ such that $\prod_{i=0}^{n} f_{r,Q_i}(P_i) \cdot u = c^r$.*

Proof. When $\prod_{i=0}^{n} e(Q_i, P_i) \neq 1$ then $\prod_{i=0}^{n} f_{r,Q_i}(P_i) = w^i \cdot s^r$ for some $s \in \mathbb{F}_{q^k}^*$ where $i \neq 0$ and w is r-th root of unity in $\mathbb{F}_{q^k}^*$. Suppose the opposite, there is a $c \in \mathbb{F}_{q^k}^*$ such that $w^i \cdot s^r \cdot u = c^r$. Then $w^i = (\frac{c}{s \cdot a})^r$, from the definition of r-rh root of unity $(w^i)^r = 1$. This implies that $(\frac{c}{s \cdot a})^{r^2} = 1$, so $ord(\frac{c}{s \cdot a}) = r^2$ which is contradiction since $r^2 \nmid q^k - 1$. □

We know that result of multiplying two non quadratic residues will be a quadratic residue [IR90, Proposition 5.1.2]. This follows from multiplicative property of Legendre symbol and can be generalized from quadratic to d-th residues [CL69] [DD95]. As noted, Miller loop outputs are always $\frac{\lambda}{d}$ residues. So in order to make them full λ residues we need to make them d-th residues. This can be done by finding a proper non d-th residue a and multiplying Miller loop output with a will make it a d-th residue. More specifically, if b is non d-th residue, then $b^{(q-1)/d} = w^i$ and $(w^i)^d = 1$. Therefore the goal is to find a such that $a^{(q-1)/d} = w^{-i}$ then $b \cdot a$ will be a d-th residue. We prove that there always exists such a and that it is additionally r-th residue, which by lemma 1 preserves the soundness.

Lemma 2. *Let b be a non d-th residue. Then there always exists a such that it is non d-th residue but it is r-th residue and $a \cdot b$ is a d-th residue.*

Proof. Let $h = d^s \cdot f$ such that $d \nmid f$. Let a be an element such that $ord(a) = d^s$. Since r is prime, $\gcd(d^s, r) = 1$ thus a is clearly r-th residue. To see that a is non d-th residue, simply suppose the opposite, i.e., there is b such that $b^d = a$. Since $ord(a) = d^s$ this would imply that $ord(b) = d^{s+1}$ which is a contradiction since $d^{s+1} \nmid h$ and thus $d^{s+1} \nmid q^k + 1$.

Since b is non d-th residue $b^{(q^k-1)/d} = w^i$ for some $i \neq 0$ and $(w^i)^d = 1$. Further since a is also non d-th residue, then all a^i for $i \in [1, d-1]$ are also non d-th residues. Therefore computing $(a^i)^{(q^k-1)/d}$ for $i \in [1, d-1]$ will be a permutation of d-th roots of unity and thus for some j it will hold that $(a^j)^{(q^k-1)/d} = w^{-i}$, thus a^j will be non d-th residue used for scaling the Miller loop output. □

From this we state the final and most important theorem of our paper which shows that testing the equality of pairings can be reduced to potentially scaling the Miller loop output with d^i-th root of unity and then checking that it is λ-residue.

Theorem 3.

$$\prod_i e(P_i, Q_i) = 1 \iff \exists c, u \in \mathbb{F}_{q^k}^* : \prod_i f_{r,Q_i}(P_i) = c^{\lambda} u \wedge u^{d^i} = 1$$

Proof. To show this, it suffices to show that there exists an x if and only if there exists c, u such that $x^r = c^\lambda u$ and $u^{d^i} = 1$.

To simplify, since $\gcd(d, rm) = 1$ there exists $v = u^{(rm)^{-1} \mod d^i}$ such that $v^{rm} = u$. There also exists $m' = m^{-1} \mod q^k - 1$, so we can rewrite the equation as $x^{rm'} = (c^d v)^{rmm'}$. Letting $y = x^{m'}$, and noting that $x = y^m$, we need to show there exists a y iff there exists c, v such that $y = c^d v$ where $v^{d^i} = 1$.

Clearly, if there exists c, v then there exists y. The decomposition of y into such a c and v follows from the Tonelli-Shanks lemma [Ton91]. Thus, given x we can compute c and v from $x^{m'}$ using Tonelli-Shanks and let $u = v^{rm}$. Given c, u we can let $x = (c^d u^{(rm)^{-1} \mod d^i})^m$. □

There is an additional optimization that we can exploit to simplify the final protocol. If the d^i primitive roots of unity lie in a proper subfield of $\mathbb{F}_{q^k}$, then we can omit the check that $u^{d^i} = 1$. The reason for this is that all elements of all subfields are always r residues, since $r \mid \Phi_k(q)$ by assumption. For many curves where d and i are small, checking u^{d^i} is an insignificant cost, but in general it may not be.

4.3 Computing Residue Witness for the BN254 curve

In this section we describe how to compute a witness c over the BN254[bn] curve. We also describe how to compute auxiliary witness when using the 'optimal' exponentiation by λ.

4.3.1 Parameters

The BN[BN05] family of curves is parametrized by the polynomials

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

It has embedding degree 12, which means $r(x) \mid q(x)^4 - q(x)^2 + 1$. Valid BN curves occur for $x_0 \in \mathbb{Z}$ where both $q = q(x_0)$ and $r = r(x_0)$ are prime. The curve currently implemented for Ethereum smart contracts is a BN curve with $x = 4965661367192848881$[alt], commonly called BN254.

For BN254, we will further define the following variables

$$\begin{aligned} h &= \frac{q^{12} - 1}{r} = 3^3 \cdot l \text{ where } \gcd(l, 3) = 1 \\ \lambda &= 6x + 2 + q - q^2 + q^3, \\ m &= \frac{\lambda}{r}, \\ d &= \gcd(m, h) = 3, \\ m' &= \frac{m}{d}. \end{aligned}$$

Equivalently $\lambda = 3rm'$. Since $\gcd(\lambda, q^{12} - 1) = 3r$, for a Miller loop output f it is always possible to find a value c such that $f = c^\lambda$, the best we can do is find some u a cubic non-residue and c such that $f = c^\lambda u$. However, since $3^3 \mid q^{12} - 1$ we need to use a modified Tonelli-Shanks algorithm to compute c.

4.3.2 Finding c

We split this computation into three parts.

1. Compute r-th root
2. Compute m'-th root
3. Compute cubic root

From section 4.2.2 we know that the Miller loop output will always be r-th and m'-th residue but it might not be a cubic residue. Therefore, before computing cube root we have to make sure that either the Miller loop output is a cubic residue or we have to multiply it with a proper non cubic residue as in lemma 2 to make it one.

Computing r-th Root. Recall that $r^2 \nmid q^12 - 1$ and equivalently $\gcd(r, h) = 1$. Therefore r has a well defined inverse mod h, which we can compute $r'r = 1 \mod h$. If $f = c^r$ for some c, then $f^h = c^{rh} = 1$. Let $c = f^{r'}$ and observe that $c^r = f^{rr'} = f^{1+hs} = f$ where $rr' = 1 + hs$. This is analogous to computing square roots when $p = 3 \mod 4$ [AGO24].

Computing m'-th Root. Recall that $\gcd(m', q^{12} - 1) = 1$ by construction. Therefore, we can use exactly the same technique to compute the m' root of f. Specifically, let $m''m' = 1 \mod q^{12} - 1$ and observe that therefore raising to the m'' is the inverse of raising to the m'. Therefore, the m' root of any element is the m'' exponentiation.

Scaling. Before proceeding to computation of cube roots we have to ensure that field element that we are working with is indeed a cubic residue. In other words, we can find a cube root of f only if there is some c such that $c^3 = f$. If f is indeed a cubic residue, then $f^{(q^k-1)/3} = 1$, otherwise $f^{(q^k-1)/3} = w^i$ where w^i is a non-trivial 3-rd root of unity. From lemma 2 we saw that there is always a proper cubic non residue a such that $(f \cdot a)^{(q^k-1)/3} = 1$. Again from lemma 2 we know that any 27-th root of unity will be non cubic residue (since $h = 27 \cdot l$ and $3 \nmid l$). Thus, for fixed 27-th root of unity a, if f is not a cubic residue then $f \cdot a$ or $f \cdot a^2$ is. Finally multiplying Miller loop output by either a or a^2 will preserve soundness since again by lemma 2 a is r-th residue.

Computing Cube Root. Computing a cube root is a slightly more complex task since $\gcd(3, h) \neq 1$. Intuitively this is precisely analogous to computing square roots when $p = 1 \mod 2^i$ for $i > 1$. Therefore to compute a cubic roots we modify Tonelli-Shanks algorithm for square roots [Ton91].

Lemma 3. *With the definitions given above, given a cubic non-residue w, calculating the x and t until $t = 0$ gives a polynomial time algorithm for computing the cube root.*

Proof. Proof is identical to proof of Tonelli-Shanks algorithm for square roots [Ton91] when powers of 2 are replaced with powers of 3 □

As in previous sections denote r' to be $r^{-1} \mod h$ and m'' to be $\frac{1}{m'} \mod h$. Then this all together gives the following steps for finding c.

Algorithm 4: Modified Tonelli-Shanks for cube roots

Input: Cube residue a, cube non residue w and write $p-1 = 3^r \cdot s$ such that $3 \nmid s$
Output: x such that $x^3 = a$

$\mathbf{exp} = (s+1)/3$
$x \leftarrow a^{\mathbf{exp}}$
$3^t \leftarrow ord(\frac{x^3}{a})$
while $t \neq 0$ **do**
  $\mathbf{exp} = (s+1)/3$
  $x \leftarrow x \cdot w^{\mathbf{exp}}$
  $3^t \leftarrow ord(\frac{x^3}{a})$
end
return x

Algorithm 5: Algorithm for computing λ residues over BN curve

Input: Output of a Miller loop f and fixed 27-th root of unity w
Output: (c, w^i) such that $c^\lambda = f \cdot w^i$

$s = 0$
if $f^{(q^k-1)/3} = 1$ **then**
  continue
end
else if $(f \cdot w)^{(q^k-1)/3} = 1$ **then**
  $s = 1$
  $f \leftarrow f \cdot w$
end
else
  $s = 2$
  $f \leftarrow f \cdot w^2$
end
$c \leftarrow f^{r'}$
$c \leftarrow c^{m''}$
$c \leftarrow c^{1/3}$ **(by using modified Tonelli-Shanks 4)**
return (c, w^s)

4.3.3 Counting Operations

In this section we outline the amount of in circuit $\mathbb{F}_{q^{12}}$ operations needed for computing c^{λ}. First of all note that finding c with Algorithm 5 is happening outside of the circuit and therefore operations needed to compute it are not counted.

Recall that $\lambda = 6x+2+q-q^2+q^3$ and that c^q, c^{q^2} and c^{q^3} can be computed in just a few $\mathbb{F}_q^2$ operations 3.4.1. c^{-q^2} is just inversion and it can be done by just one multiplication inside the circuit. Thus the exponentiation c^{λ} is dominated by exponentiation c^{6x+2}. Another important observation that we make is that the exponentiation c^{6x+2} can be embedded into computation of the Miller loop and thus all squarings are essentially 'free'. Then writing $6x+2$ in the form $\sum s_i 2^i$ for $s_i \in \{-1, 0, 1\}$ has Hamming weight 26 and together with inverse check the exponentiation c^{6x+2} requires only 27 $\mathbb{F}_{q^{12}}$ multiplications.

5 Precomputing Lines

A significant part of Miller function evaluation consists of line constructions and evaluations. Lines are constructed from coordinates of points they pass through. A natural question that arises is the choice of the coordinate system in which points should be represented. Pairings in general can be computed over points represented in any coordinate system but depending on a ratio between inversions and multiplications, the most popular choices are homogeneous projective and affine coordinate systems. We show that one of the most optimal in circuit computation is obtained from using affine coordinate system when instead of constructing line through T, Q $\mathcal{P}$ additionally witnesses line coefficients (λ, μ) and proves that those coefficients indeed define a line that passes through T, Q. We then observe that, when point $Q \in \mathbb{G}_2$ at which pairing is computed is fixed, all line coefficients can be precomputed and hardcoded inside the circuit. We assume that extension fields are constructed from towers [BS09] and we outline the results for D-type twists. A similar approach can be applied to M-type twists.

5.1 Sparse $\mathbb{F}_{q^k}$ multiplication

Barreto *et al.* [BKLS02] observed that $q-1|(q^k-1)/r$, thus multiplications by elements in $\mathbb{F}_q$ do not change the result of pairing, i.e. scaling elements from $\mathbb{F}_{q^k}$ by arbitrary $x \in \mathbb{F}_q$ doesn't affect pairing output since after final exponentiation $x^{(q^k-1)/r} = 1$. Aranha *et al.* [ABLR13] used this observation to show that line evaluations can be scaled by an element from $\mathbb{F}_q$ to obtain a sparse element in $\mathbb{F}_{q^k}$. More specifically for sextic twists, normalized line evaluation will output an element of the form $(1 + g_1 w)$. Then computing $(f_0 + f_1 w) \cdot (1 + g_1 w)$ requires less multiplications compared to multiplying $(f_0 + f_1 w) \cdot (g_0 + g_1 w)$ for $g_0 \neq 1$. We utilize the same optimization, and we compare our result with results in Aranha *et al.* [ABLR13] for normalized lines.

5.2 Costs

By $\tilde{m}, \tilde{s}, \tilde{a}, \tilde{\imath}$ denote multiplication, squaring, addition and inversion in $\mathbb{F}_{q^{(k/d)}}$ respectively and by m denote multiplication of element in $\mathbb{F}_{q^{(k/d)}}$ by element in $\mathbb{F}_q$. Then Aranha *et al.* [ABLR13] obtains the following results:

Affine. After precomputation of $x'_P = \frac{-x_P}{y_P}, y'_P = \frac{1}{y_P}$, where $P = (x_P, y_P)$ is the point at which lines are evaluated, doubling and addition operations together with line evaluation can be computed in $3\tilde{m} + 2\tilde{s} + 7\tilde{a} + \tilde{\imath} + 4m$ and $3\tilde{m} + \tilde{s} + 6\tilde{a} + \tilde{\imath} + 4m$ respectively and line evaluation $l_{T,Q}(P)$ equals to $1 + \lambda x'_P w + y'_P(\lambda x_1 - y_1)w^3$ where λ is the coefficient of line through T, Q.

Homogeneous. After precomputation of $\overline{x}_P = -x_P, \overline{y}_P = -y_P, x'_P = 3x_P$, where $P = (x_P, y_P)$ is again the point at which lines are evaluated, doubling and addition operations together with line evaluation can be computed in $3\tilde{m}+6\tilde{s}+16\tilde{a}+4m$ and $11\tilde{m}+2\tilde{s}+8\tilde{a}+4m$ respectively.

Since in circuit computation of inverse can be replaced with a single multiplication, it is clear that affine representation is a better choice. We further proceed in showing that by witnessing (λ, μ) we can save additional $\tilde{m}$ and m operations.

Double and evaluate. Given a point $T = (x_1, y_1) \in E(\mathbb{F}_{q^{(k/d)}})$ and $P \in E(\mathbb{F}_q)$ the task is to compute $(x_3, y_3) = T + T$ and $g = l_{T,T}(P)$. After the same precomputation of x'_P, y'_P naive application of results obtained in Aranha *et al.* [ABLR13] will be dominated by $4\tilde{m}$ and one $2\tilde{s}$ (since $\tilde{\imath} = \tilde{m}$ in circuit). By additionally witnessing (λ, μ) one $\tilde{m}$ and $2m$ operations can be saved in the following way.

1. Show that the pair (λ, μ) indeed define a tangent through T showing that $y_1 - \lambda x_1 - \mu = \mathcal{O}$ and $2\lambda y_1 = 3{x_1}^2$. This step is dominated by $2\tilde{m}$ and one $\tilde{s}$.
2. Compute λ^2 which is simply one $\tilde{s}$
3. Compute $x_3 = \lambda^2 - 2x_1$ and $y_3 = -\mu - \lambda x_3$ which is dominated by computing λx_3, i.e. $1\tilde{m}$
4. Compute $l_{T,Q}(P) = 1 + \lambda x'_P - y'_P \mu w^3$ which requires $2m$ operations.

Add and evaluate. Given points $T = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_{q^{(k/d)}})$ and $P \in E(\mathbb{F}_q)$ the task is to compute $(x_3, y_3) = T + Q$ and $g = l_{T,Q}(P)$. After the same precomputation of x'_P, y'_P naive application of results obtained in [ABLR13] will be dominated by $4\tilde{m}$ and one $\tilde{s}$ (since $\tilde{\imath} = \tilde{m}$ in circuit). By additionally witnessing (λ, μ) one $\tilde{m}$ and $2m$ operations can be saved in the following way.

1. Show that the pair (λ, μ) indeed define a line through T and Q by showing that $y_1 - \lambda x_1 - \mu = \mathcal{O}$ and $y_2 - \lambda x_2 - \mu = \mathcal{O}$. This step is dominated by $2\tilde{m}$.
2. Compute λ^2 which is simply one $\tilde{s}$
3. Compute $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = -\mu - \lambda x_3$ which is dominated by computing λx_3, i.e. $1\tilde{m}$
4. Compute $l_{T,Q}(P) = 1 + \lambda x'_P - y'_P \mu w^3$ which requires $2m$ operations.

We proceed to define a line object and it's functionalities that we utilize in the Miller loop computation. In the Algorithm 6 we outline all the steps of modified Miller loop that takes line coefficients as an input. Then we argue further optimizations where Miller loop is computed at predetermined point Q. For this purpose we define an indexer with the Algorithm 7 and another modification of Miller loop with Algorithm 8

Line object Define a '*line*' object that is determined by $(\lambda, \mu) \in \mathbb{F}^2_{q^{(k/d)}}$ and has the following functionalities:

1. $line.is\_tangent(T) :: E(\mathbb{F}_{q^{(k/d)}}) \to bool$: returns true if line is a tangent through point T
2. $line.is\_line(T, Q) :: E(\mathbb{F}_{q^{(k/d)}}) \times E(\mathbb{F}_{q^{(k/d)}}) \to bool$: returns true if line passes through points (T, Q)
3. $line.evaluate(P) :: E(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^{(k/d)}}$: evaluates line in P
4. $line.double(T) :: E(\mathbb{F}_{q^{(k/d)}}) \to E(\mathbb{F}_{q^{(k/d)}})$: given point T returns $2T$
5. $line.add(T, Q) :: E(\mathbb{F}_{q^{(k/d)}}) \times E(\mathbb{F}_{q^{(k/d)}}) \to E(\mathbb{F}_{q^{(k/d)}})$: given points T and Q returns $T + Q$

Algorithm 6: Miller loop with precomputed lines

Input: $Q \in E(\mathbb{F}_{q^{(k/d)}}), P \in E(\mathbb{F}_q), \mathbf{L} := [line(\lambda_0, \mu_0), line(\lambda_1, \mu_1), \ldots, line(\lambda_{n-1}, \mu_{n-1})], e = \sum_{i=0}^{L-1} s_i 2^i$ for $s_i \in \{-1, 0, 1\}$ and $n = L - 1 + \sum_{i=0}^{L-2} |s_i|$

Output: $f_{e,Q}(P)$

$T \leftarrow Q, f \leftarrow 1$
$lc \leftarrow 0$
for $i = L - 2$ **to** 0 **do**
  $l \leftarrow \mathbf{L}[lc], lc \leftarrow lc + 1$
  assert $l.is\_tangent(T)$
  $f \leftarrow f^2 \cdot l.evaluate(P)$
  $T \leftarrow l.double(T)$
  if $s_i^2 = 1$ **then**
    $Q' \leftarrow Q$ **if** $s_i = 1$ **else** $-Q_j$
    $l \leftarrow \mathbf{L}[lc], lc \leftarrow lc + 1$
    assert $l.is\_line(T, Q')$
    $f \leftarrow f \cdot l.evaluate(P)$
    $T \leftarrow l.add(T, Q')$
  end
end
return f

5.3 Predefined Q

Protocol such as Plonk or more specifically KZG polynomial commitment scheme is verified by checking equality of pairings in predetermined fixed points $Q_1, Q_2 \in \mathbb{G}_2$. We observe that in this setting all line coefficients (λ_i, μ_i) together with all accumulation point T coordinates do not depend on P. Thus we delegate this computation to indexer $\mathcal{I}$ (7) that computes all lines and points which are then hardcoded into the circuit. This is one time computation that is also being performed by the verifier, thus checking correctness of lines can be fully omitted from Miller loop computation (8).

5.4 Counting Operations

We show that affine coordinate representation is more suitable for in circuit computations and that additional optimizations can further reduce curve operations. Further we show that when computing pairings in predefined points all computations depending on lines can be reduced only to $2m$ operations, i.e.line evaluations.

Bellow we count the asymptotic number of multiplications needed to compute a Miller loop. Let **sk** denote squaring of full $\mathbb{F}_{q^k}$ element and $\tilde{spk}$ denote multiplication of one full $\mathbb{F}_{q^k}$ element with one sparse $\mathbb{F}_{q^k}$ element (that is in general obtained from normalized line evaluation). Without loss of generality we assume that accumulator f in Miller loop is always a full field element in $\mathbb{F}_{q^k}$. Operations are counted for computing $f_{e,Q}(P)$ where $e = \sum_{i=0}^{L-1}$ for $s_i \in \{-1, 0, 1\}$ and $s_{L-1} = 1$. Using $\{-1, 0, 1\}$ representation achieves smaller or equal Hamming weight compared to $\{0, 1\}$ representation, thus requires less multiplications overall. Accumulator f is squared and scaled by line evaluation output at each of $L - 1$ steps of the loop and additionally scaled by another line evaluation whenever $s_i \in \{-1, 1\}$. Additionally at each of $L - 1$ steps a tangent line

Algorithm 7: Indexer for fixed point Q

Input: $Q \in E(\mathbb{F}_{q^{(k/d)}})$ and $e = \sum_{i=0}^{L-1} s_i 2^i$ for $s_i \in \{-1, 0, 1\}$
Output: $\mathbf{L}$

$\mathbf{L} \leftarrow []$
$T \leftarrow Q$
for $i = L - 2$ **to** 0 **do**
  $\mathbf{L}.push(line(T, T))$
  $T \leftarrow 2T$
  if $s_i^2 = 1$ **then**
    $\mathbf{L}.push(line(T, s_i Q))$
    $T \leftarrow T + s_i Q$
  end
end
return $\mathbf{L}$

Algorithm 8: Miller loop in fixed Q

Input: $Q \in E(\mathbb{F}_{q^{(k/d)}}), P \in E(\mathbb{F}_q), \mathbf{L} := \mathcal{I}(Q)$ and $e = \sum_{i=0}^{L-1} s_i 2^i$ for $s_i \in \{-1, 0, 1\}$
Output: $f_{e,Q}(P)$

$f \leftarrow 1$
$lc \leftarrow 0$
for $i = L - 2$ **to** 0 **do**
  $l \leftarrow \mathbf{L}[lc]$
  $f \leftarrow f^2 \cdot l.evaluate(P)$
  if $s_i^2 = 1$ **then**
    $l \leftarrow \mathbf{L}[lc + 1]$
    $f \leftarrow f \cdot l.evaluate(P)$
  end
  $lc \leftarrow lc + 2$
end
return f

is constructed from accumulator point T, that line is evaluated at P and T is doubled. Then, whenever $s_i \in \{-1, 1\}$ another line is constructed from T, s_iQ, that lines is evaluated at P and T is set to be $T + s_iQ$. Let A be $\sum_{i=0}^{L-2} |s_i|$ and $\tilde{m}, \tilde{s}$ and m be defined as in 8.2, then we obtain following operation count.

Affine. At each step f is squared and scaled with sparse l that is obtained from line evaluation. This together produces $(L-1) \cdot (\mathbf{sk} + \mathbf{spk})$ operations. Further whenever $s_i \in \{-1, 1\}$ f is scaled by another sparse element, which produces $A \cdot \mathbf{spk}$ operations. Checking that pair (λ, μ) defines a tangent line through T takes $3\tilde{m} + \tilde{s}$ operations and doubling T takes $\tilde{m} + \tilde{s}$ operations. Evaluating line in P takes $2m$ operations and in total it adds to $(L-1) \cdot (4\tilde{m} + 2\tilde{s} + 2m)$. Similarly when $s_i \in \{-1, 1\}$ checking that pair (λ, μ) defines a line that passes through (T, s_iQ) takes $2\tilde{m}$ and updating T takes $\tilde{m} + \tilde{s}$ operations. Finally evaluating line at P again takes $2m$ operations which in total adds to $A \cdot (3\tilde{m} + \tilde{s} + 2m)$. Note that this operations correspond to computing $\mathit{line.is\_tangent}(T), \mathit{line.double}(T), \mathit{line.is\_line}(T, s_iQ)$ and $\mathit{line.add}(T, s_iQ)$ accordingly.

Affine with predefined Q. When Q is predetermined then all lines can be precomputed in the indexing phase (7) and hardcoded into the circuit. Then both checking correctness of lines and updating T can simply be omitted, which removes all $\tilde{m}$ and $\tilde{s}$ operations. Again, at each step f is squared and scaled with sparse l that is obtained from line evaluation. This together produces $(L-1) \cdot (\mathbf{sk} + \mathbf{spk})$ operations. Also for $s_i \in \{-1, 1\}$ scaling f adds $A \cdot \mathbf{spk}$ operations. Main optimization follows from the fact that $\mathcal{P}$ only needs to evaluate lines at P which is produces $(L - 1 + A) \cdot 2m$ operations.

Following table captures full operation count:

Q predefined	$\mathbb{F}_{q^k}$ operations	$E(\mathbb{F}_{q^{(k/d)}})$ operations
false	$(L-1) \cdot (\mathbf{sk} + \mathbf{spk}) + A \cdot \mathbf{spk}$	$(L-1) \cdot (4\tilde{m} + 2\tilde{s} + 2m) + A \cdot (3\tilde{m} + \tilde{s} + 2m)$
true	$(L-1) \cdot (\mathbf{sk} + \mathbf{spk}) + A \cdot \mathbf{spk}$	$(L - 1 + A) \cdot 2m$

Table 1: Miller loop operation count

6 Randomized Field Arithmetic

Let $\mathbb{F}$ be a field and let $p(X) \in \mathbb{F}[X]$ be an irreducible polynomial. Suppose that $\mathbb{K}$ is extension of $\mathbb{F}$ containing root of $p(X)$, α. Denote by $\mathbb{F}(\alpha)$ subfield of $\mathbb{K}$ generated by α over $\mathbb{F}$. Then

$$\mathbb{F}(\alpha) \cong \mathbb{F}[X]/(p(X))$$

Further, this isomorphism is induced by mapping[ext]:

$$\varphi : \mathbb{F}[X]/(p(X)) \to \mathbb{F}(\alpha)$$

This essentially means that arithmetic over $\mathbb{F}(\alpha)$ can be treated as arithmetic over $\mathbb{F}[X]$ modulo irreducible polynomial $p(X)$. Showing that $a \cdot b = c$ for $a, b, c \in \mathbb{F}(\alpha)$ is equivalent to showing that there exist a unique polynomial $q(X)$ such that

$$a(X)b(X) - c(X) = q(X)p(X)$$

Where $a(X), b(X), c(X) \in \mathbb{F}[X]/(p(X))$ are respective polynomial representations of a, b, c induced by φ. Then testing this polynomial identity can be reduced to testing it in a random point by Schwartz-Zippel lemma, which is also the approach employed by Garaga [gar]. More generally

testing that some relation $\mathcal{R}$ holds for some $a_1, a_2, \dots a_n \in \mathbb{F}(\alpha)$ can be reduced to testing that there exist a unique polynomial $q(X)$ such that

$$\mathcal{R}(a_1(X), a_2(X), \dots, a_n(X)) = q(X)p(X)$$

In Garaga [gar] each $\mathbb{F}_{q^k}$ multiplication is replaced by one polynomial identity check of the form:

$$a_i(X)b_i(X) - c_i(X) = q_i(X)p(X)$$

This requires prover to commit and evaluate a quotient polynomial $q_i(X)$ for each multiplication. We propose an optimization where by introducing another round of interaction prover commits to only one quotient $Q(X)$ across all $\mathbb{F}_{q^k}$ multiplications. More specifically prover first commits to all witness polynomials $a_i(X), b_i(X), c_i(X)$. Verifier then sends a separation challenge β and prover then sends one quotient polynomial $Q(X)$ claimed to satisfy relation:

$$\sum_{i=0}^{n} \beta^i (a_i(X)b_i(X) - c_i(X)) = Q(X)p(X)$$

Verifier then samples a random evaluation challenge γ at which this relation is then checked. Checking m relations $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_m$ spanned across n polynomials $w_1(X), w_2(X), \dots, w_n(X) \in \mathbb{F}[X]/(p(X))$ can be captured by the following interactive argument:

Extension Field Arithmetic IOP

1. $\mathcal{P}$ commits to $w_1(X), w_2(X), \dots, w_n(X) \in \mathbb{F}[X]/(p(X))$
2. $\mathcal{V}$ selects β
3. $\mathcal{P}$ commits to $Q(X) = (\sum_{j=1}^{m} \beta^j \mathcal{R}_j(w_1(X), w_1(X), \dots, w_n(X)))/p(X)$
4. $\mathcal{V}$ selects α
5. $\mathcal{P}$ shows that $w_{i,\alpha} = w_i(\alpha)$ and $\sum_{j=1}^{m} \beta^j \mathcal{R}_j(w_{1,\alpha}, w_{2,\alpha}, \dots, w_{n,\alpha}) = Q(\alpha)p(\alpha)$

Garaga [gar] samples an evaluation challenge by hashing the witness inside the circuit. Instead, when the proof system is derived from an interactive protocol (using the Fiat-Shamir transform), we can modify the underlying protocol such that the verifier provides the additional randomness [cha] and thus remove the need of in-circuit hashing.

Efficient Polynomial Evaluation. By committing all $w_i(X)$ column-wise efficient computation of $w_{i,\alpha}$ for all $w_i(X)$ can be done in the following way. Let d be the degree of all $w_i(X) \in \mathbb{F}[X]/(p(X))$ and write $w_i(X) = w_{i,d}X^d + w_{i,d-1}X^{d-1} +, \cdots + w_{i,0}$. Assume that $n \cdot (d+1)$ is power of two. By $\mathbb{H}$ denote a multiplicative subgroup of $\mathbb{F}$ of size $n \cdot (d+1)$ generated by $g \in \mathbb{F}$ and by $z_{\mathbb{H}}$ denote a vanishing polynomial of $\mathbb{H}$. Further prover and verifier agree and precompute polynomial $h(X)$ such that:

$$\begin{cases} h(g^i) = 1 & \text{when } i \equiv 0 \mod d+1 \\ h(g^i) = 0 & \text{otherwise} \end{cases}$$

By $\mathbf{w}_i$ denote array of coefficients $[w_{i,d}, w_{i,d-1}, \dots w_{i,0}]$. Then prover first commits to a polynomial $c(X)$ which evaluations are constructed by concatenation of all $\mathbf{w}_i$ arrays. Verifier

then sends an evaluation challenge α and prover commits to polynomial $e(X)$. Prover and verifier then engage in an interactive argument for checking that

$$h(X)(c(X) - e(X)) = q_1(X)z_{\mathbb{H}}(X) \tag{1}$$

$$(1 - h(X))(e(X) - (e(g^{-1}X)\alpha + c(X))) = q_2(X)z_{\mathbb{H}}(X) \tag{2}$$

The table 2 outlines polynomials $h(X), c(X), e(X)$ evaluations for $n = 2, d = 3$:

Table 2: Column-wise efficient evaluation

$h(X)$	$c(X)$	$e(X)$
1	$w_{0,3}$	$w_{0,3}$
0	$w_{0,2}$	$w_{0,3}\alpha + w_{0,2}$
0	$w_{0,1}$	$(w_{0,3}\alpha + w_{0,2})\alpha + w_{0,1}$
0	$w_{0,0}$	$((w_{0,3}\alpha + w_{0,2})\alpha + w_{0,1})\alpha + w_{0,0} = w_0(\alpha)$
1	$w_{1,3}$	$w_{1,3}$
0	$w_{1,2}$	$w_{1,3}\alpha + w_{1,2}$
0	$w_{1,1}$	$(w_{1,3}\alpha + w_{1,2})\alpha + w_{1,1}$
0	$w_{1,0}$	$((w_{1,3}\alpha + w_{1,2})\alpha + w_{1,1})\alpha + w_{1,0} = w_1(\alpha)$

Thus when underlying proof system is an IOP[BSCS16] all extension field arithmetic can be checked with the following IOP

Extension Field Arithmetic IOP

1. $\mathcal{P}$ commits to $c(X)$
2. $\mathcal{V}$ selects β
3. $\mathcal{P}$ commits to $Q(X) = (\sum_{j=1}^{m} \beta^j \mathcal{R}_j(w_1(X), w_1(X), \ldots, w_n(X)))/p(X)$
4. $\mathcal{V}$ selects α
5. $\mathcal{P}$ sends commitment to $e(X)$ and opening $Q(\alpha)$
6. $\mathcal{V}$ selects γ
7. $\mathcal{P}$ commits to $B(X) = \frac{h(X)(c(X)-e(X))+\gamma((1-h(X))(e(X)-(e(g^{-1}X)\alpha+c(X))))}{z_{\mathbb{H}}(X)}$
8. $\mathcal{V}$ selects v
9. $\mathcal{P}$ sends openings $B(v), h(v), c(v), e(v), e(g^{-1}v)$
10. $\mathcal{V}$ checks that $\sum_{j=1}^{m} \beta^j \mathcal{R}_j(w_{1,\alpha}, w_{2,\alpha}, \ldots, w_{n,\alpha}) = Q(\alpha)p(\alpha)$
11. $\mathcal{V}$ checks that $h(v)(c(v) - e(v)) + \gamma((1 - h(v))(e(v) - (e(g^{-1}v)\alpha + c(v)))) = B(v)z_{\mathbb{H}}(v)$

6.1 Compressing the Miller Loop

Recall that the Miller loop is essentially computed as a double-and-add addition chain. At each step we square an accumulator, evaluate some lines, and multiply the evaluations with the

accumulator. When encoded directly into a SNARK, the prover must commit to the accumulator at the end of every step.

We could however "compress" multiple rounds of the Miller loop into a single computation. For example, suppose two consecutive bits of the Miller loop were zero and instead of two squarings we checked $f_{i+2}(P) = f_i(P)^4 l_i(P)^2 l_{i+1}(P)$. That is, we do not commit to the value $f_{i+1}(P) = f_i(P)^2 l_i(P)$. Clearly, this is sound but when working with plain $\mathbb{F}_{q^k}$ arithmetic it dramatically increases the size of the multiplication circuit. Heuristically, without committing to intermediate results, the number of products of $\mathbb{F}$ elements that we must compute grows exponentially in the degree.

However, using randomized field arithmetic there is no such exponential increase in complexity, we just increase the degree of the final quotient. In this case, the quotient degree becomes $7(k-1)-k$ assuming we commit to f_i and the line coefficients. That is, we add $6k-3$ to the quotient *independent of the number of rounds of the Miller loop.* Compressing d rounds increases the cost by a factor of $2^d - 1$, so clearly there is a limit to how much we can use this trick. When measured in commitment costs, the break even point can be calculated directly from the number of rounds of the Miller loop. For L rounds, we need the value of d which minimizes

$$C = \min_d (k-1)L/d + (2^d - 1)(k-1) - k$$

When the $s_i \neq 0$, a finer analysis is needed.

6.2 Replacing Lines with Divisors

Of the $2^d - 1$ increase in degree, $2^{d-1} - 1$ of it comes from lines, and potentially more when $s_i \neq 0$. To remedy this, we can combine the lines into a higher degree divisor. Consider a selection of d digits $s_i, \ldots s_{i+d-1}$ and let $t_i = \sum_{i=0}^{d-1} s_i 2^i$. Suppose the accumulator before bit performing these rounds of the miller loop is T_i, and after it is $T_{i+1} = T_i + \sum_{i=0}^{d-1} s_i Q$. We define the modified divisor

$$\mathrm{div}(g_i) = t_i(Q) + (T_{i+1}) + (-T_i) + \sum_{i=0}^{d-1} s_i(2^i Q) - n_i(\mathcal{O}).$$

Simple arithmetic shows that this divisor is indeed principal. Note that multiplying by this divisor has the effect of multiplying and squaring all the lines in the Miller loop. When written as a polynomial $g_i(X, Y) = a_i(X) + Y b_i(X)$ the number of coefficients to express this divisor is $n_i < 2^{d+2} + 1$ and in general depends on how many $s_i = 0$. Using wNAF, we expect that at most half the digits are a non-zero, so $n_i < 2^{d+1} + 1$.

This again introduces an interesting trade off between the cost of committing to more quotient degree terms and committing to larger divisors. In the case of fixed G_2 inputs, we can pre-commit to all the divisors which favors using larger d. However, it will increase the number of gates past a certain point.

This raises an interesting issue: how do we verify that the divisors are correct in the case with the G_2 argument is not fixed. We believe that it may be possible to adapt the protocol of Eagen [Eag22], but leave this for future work and only propose this optimization for the fixed G_2.

6.3 Alternative Bases

Efficiency of pairing computation heavily depends on the complexity of $\mathbb{F}_{q^k}$ arithmetic. Working over k degree irreducible polynomials and representing elements with tuples of k elements would require k^2 $\mathbb{F}_q$ operations to perform a single multiplication. Therefore the key motivation for

using towers is to exploit the fact that there are more efficient methods for computing operations in $\mathbb{F}_{q^2}$ and $\mathbb{F}_{q^3}$. Specifically, by utilizing the Karatsuba method[KO62] one can compute $\mathbb{F}_{q^2}$ multiplication using 3 $\mathbb{F}_q$ multiplications and squaring in $\mathbb{F}_{q^2}$ by using 2 $\mathbb{F}_q$ multiplications. Similar optimizations exists for cubic extensions, and therefore multiplications in $\mathbb{F}_{q^k}$ can be done in significantly less than k^2 $\mathbb{F}_q$ multiplications. Further, in the section 3.4.1 we showed that Frobenius operator can also be efficiently computed when using towering. Thus when computing pairings tower representation of $\mathbb{F}_{q^k}$ elements gives the best performance, however when using Randomized Polynomial Arithmetic multiplications in $\mathbb{F}_{q^k}$ are not explicitly computed inside the circuit. Therefore instead of using towers we can represent $\mathbb{F}_{q^k}$ elements as a coefficients in normal basis, i.e., given $x \in \mathbb{F}_{q^k}$ we represent it with array of k coefficients such that $x = \sum_{i=0}^{k-1} a_i \alpha^{q^i}$ for $a_i \in \mathbb{F}_q$ and some $\alpha \in \mathbb{F}_{q^k}$. Then computing Frobenius operator is just a cyclic shift which additionally reduces the constraints compared to approach we mentioned in 3.4.1.

7 Verifiable Pairings

In this section we outline all the steps that prover has to execute inside the circuit in order to prove equality between two products of pairings over the BN curve.

7.1 Omitting d-th residue checks

In section 4.3 we showed that, for a fixed 27-th root of unity w, prover potentially has to scale the Miller loop output by either w or w^2. Denote by s the arbitrary witness that prover uses to scale the Miller loop. To make sure that it is indeed using a correct value we can simply check that $0 \leftarrow (1-s)(w-s)(w^2-s)$. However we proceed to show that this check can be removed from the circuit. Observe that $27|q^3-1$, thus all 27-th roots of unity lie in the $\mathbb{F}_{q^3}$ which is subfield of $\mathbb{F}_{q^k}$. And by [BKLS02] and lemma 1 we know that $x \in \mathbb{F}_{q^3}$ cannot affect the pairing output. Thus, instead of constraining that $\mathcal{P}$ indeed uses a correct non-cubic residue we can simply change the circuit such that it only accepts the $\mathbb{F}_{q^3}$ elements for the scaling of the Miller loop output. The same method can be applied to other curves when there exists a non d-th residue that lie in the subfield of $\mathbb{F}_{q^k}$.

7.2 Multi Miller loop

Running multi pairing computation can be done by separately executing the Miller loop for each pair of points and then accumulating results. The very common optimization for multi pairing computation is called a multi Miller loop[Sco19]. It runs all Miller loops in parallel and at each step all line evaluations are multiplied into the same accumulator. Then when accumulator gets squared it essentially squares all separate accumulators at the same time. This significantly reduces amount of $\mathbb{F}_{q^k}$ operations compared to the straw man approach which separately squares accumulator at each step of the Miller loop for each pair of points. One of the key observations of this paper is that the exponentiation c^{6x+2} can also be embedded into the multi Miller loop and thus all squarings during this exponentiation are essentially free. We outline this in the Algorithm 9.

Given points:

$$[(P_1, Q_1), (P_2, Q_2), \ldots, (P_n, Q_n)]$$

Prover wishes to show that $\prod_{i=0}^{n} e(P_i, Q_i) = 1$. For each predefined $T \in \mathbb{G}_2$ by $\mathbf{P}_T$ denote all line coefficients that are obtained by running indexer 7. Then prover starts by preparing auxiliary witness, then it proves execution of the circuit defined with algorithm 9.

Algorithm 9: Multi Miller loop with embedded c exponentiation

Input: $\mathbf{A} = [(P_1, Q_1), (P_2, Q_2), \ldots, (P_n, Q_n)], c, c^{-1} \in \mathbb{F}_{q^k}, s \in \mathbb{F}_{q^3}, \mathbf{P}_{Q_j} \leftarrow \mathcal{I}(Q_j)$
Output: 1 if $\prod_{i=0}^{n} e(P_i, Q_i) = 1$

assert $c \cdot c^{-1} = 1$
$f \leftarrow c^{-1}, lc \leftarrow 0$
Initialize array $\mathbf{T}$ such that $\mathbf{T}[j] = Q_j$ for each non fixed point Q_j
for $i = L - 2$ **to** 0 **do**
    $f \leftarrow f^2$
    for $j = 1$ **to** n **do**
        $l \leftarrow \mathbf{P}_{Q_j}[lc]$
        $f \leftarrow f \cdot l.evaluate(P_j)$
        if Q_j *is not fixed* **then**
            $T \leftarrow \mathbf{T}[j]$
            assert $l.is\_tangent(T)$
            $\mathbf{T}[j] \leftarrow l.double(T)$
        end
        if $s_i^2 = 1$ **then**
            $f \leftarrow f \cdot c'$
            $l \leftarrow \mathbf{P}_{Q_j}[lc + 1]$
            $f \leftarrow f \cdot l.evaluate(P_j)$
            if Q_j *is not fixed* **then**
                $(Q', c') \leftarrow (Q_j, c)$ **if** $s_i = 1$ **else** $(-Q_j, c^{-1})$
                $T \leftarrow \mathbf{T}[j]$
                assert $l.is\_line(T, Q')$
                $\mathbf{T}[j] \leftarrow l.add(T, Q')$
            end
        end
    end
    $lc \leftarrow lc + 2$
    for $j = 0$ **to** n **do**
        $f \leftarrow f \cdot s \cdot (c^{-1})^q \cdot (c^{-1})^{q^2} \cdot (c^{-1})^{q^3}$
        $l_{1:3} \leftarrow (\mathbf{P}_{Q_j}[lc + i])_{i=0}^{2}$
        $f \leftarrow f \cdot l_1.evaluate(P_j) \cdot l_2.evaluate(P_j) \cdot l_3.evaluate(P_j)$
        if Q_j *is not fixed* **then**
            $Q_1 \leftarrow \pi_p(Q), Q_2 \leftarrow \pi_p(Q_1), Q_3 \leftarrow \pi_p(Q_2)$
            $T \leftarrow \mathbf{T}[j]$
            assert $l_1.is\_line(T, Q_1)$; $T \leftarrow T + Q_1$
            assert $l_2.is\_line(T, -Q_2)$; $T \leftarrow T - Q_1$
            assert $l_3.is\_line(T, Q_3)$
        end
    end
end
return $f \stackrel{?}{=} 1$

Witness preparation. If some of the $\mathbb{G}_2$ points are fixed then by design their line coefficients are already hardcoded inside the circuit. For each non predefined point $T \in \mathbb{G}_2$ prover computes required line coefficients with the Algorithm 7, denote this array by $\mathbf{L}_T$. It then proceeds with computing $f \leftarrow \prod_{i=0}^{n} f_{r,Q_i}(P_i) \cdot \prod_{j=0}^{m} f_{r,S_j}(-R_j)$. Then given f it further computes (c, w^i) with Algorithm 5.

8 On-chain verification

On-chain pairing computation can be simplified following the same approach we take for in circuit computation. By additionally submitting c, w (a proper cubic non residue power) and all line coefficients, final exponentiation can be omitted by checking $f_{r,Q}(P) \cdot w = c^r$. Naturally, in the cases where Q is predefined, such as in verifying KZG, we would also benefit from precomputation of all lines through Q. However, storage for all line coefficients on-chain is too expensive. To deal with this ambiguity we propose simply storing a verifiable merkle list commitment, cm to all line coefficients. In other words, given line coefficients $\{(\lambda_0, \mu_0), (\lambda_1, \mu_1), \ldots, (\lambda_n, \mu_n)\}$, party deploying contract can submit $cm = \mathcal{H}((\lambda_0, \mu_0), \mathcal{H}((\lambda_1, \mu_1), \ldots, \mathcal{H}((\lambda_{n-1}, \mu_{n-1}), (\lambda_n, \mu_n))))$ where $\mathcal{H}$ is chosen to be a hash function such that it is cheap to compute on-chain. Then, for every proof submitted, the contract will first compute merkle list commitment to submitted lines cm' and continue execution only if $cm' = cm$, otherwise it reverts. Suppose that party A wants to submit points $P, R \in \mathbb{G}_1$ such that $e(P, Q) = e(R, S)$ and that $Q, S \in \mathbb{G}_2$ are predetermined. Then following notation from section 7 contract stores cm_Q and cm_S that are merkle list commitments to $\mathbf{P}_Q$ and $\mathbf{P}_S$ respectively. Then the on-chain paring equality check can be performed with Algorithm 10.

Algorithm 10: Algorithm for proving equality of pairings over BN curve

Input: $P, R, c, c^{-1}, s \in \mathbb{F}_{q^3}$, two arrays of lines $\mathbf{P}'_Q, \mathbf{P}'_S$ claimed to be $\mathbf{P}_Q, \mathbf{P}_S$ respectively

Output: 1 if $e(P, Q) = e(R, S)$

1 **Compute merkle list commitments** cm'_Q **and** cm'_S **of** $\mathbf{P}'_Q, \mathbf{P}'_S$ **respectively**

2 **if** $cm'_Q \neq cm_Q$ **or** $cm'_Q \neq cm_Q$ **return** 0

3 $MillerWithC([(P, Q), (-R, S)], c, c^{-1}, s, \mathbf{P}'_Q, \mathbf{P}'_S)$ **(Algorithm 9)**

9 Conclusion

We demonstrate that verifying the equality of pairings entails a distinctly different approach compared to their computation. Most remarkably we show that the most expensive part of pairing, final exponentiation, can be replaced with only one 'short' exponentiation c^λ, where λ is the same parameter that is used for obtaining the shortest Miller loop. Further we show that in various scenarios when $\mathbb{G}_2$ points at which we compute pairings are predetermined all operations on twisted curve can be verifiably precomputed. Finally we argue that the exact same technique can be utilized to obtain cheaper on-chain pairings verification which opens a question of practicality of verifying SNARKs on Bitcoin[bitb].

9.1 Acknowledgments

The authors would like to thank Kobi Gurkan, Lai Ying Tong and Nicolas Mohnblatt for useful discussions and comments.

References

[ABLR13] Diego F. Aranha, Paulo S. L. M. Barreto, Patrick Longa, and Jefferson E. Ricardini. The realm of the pairings. Cryptology ePrint Archive, Paper 2013/722, 2013. https://eprint.iacr.org/2013/722.

[AGO24] Ebru Adiguzel-Goktas and Enver Ozdemir. Square root computation in finite fields, 2024.

[alt] https://eips.ethereum.org/EIPS/eip-1108.

[BCL+20] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. Cryptology ePrint Archive, Paper 2020/1618, 2020. https://eprint.iacr.org/2020/1618.

[BDM+10] Jean-Luc Beuchat, Jorge Enrique 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. Cryptology ePrint Archive, Paper 2010/354, 2010. https://eprint.iacr.org/2010/354.

[bita] https://bitvm.org/.

[bitb] https://bitcoin.org/bitcoin.pdf.

[BKLS02] Paulo S. L. M. Barreto, Hae Y. Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. Cryptology ePrint Archive, Paper 2002/008, 2002. https://eprint.iacr.org/2002/008.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17:297–319, 2004.

[bn] https://neuromancer.sk/std/bn/bn254.

[BN05] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. Cryptology ePrint Archive, Paper 2005/133, 2005. https://eprint.iacr.org/2005/133.

[BS09] Naomi Benger and Michael Scott. Constructing tower extensions for the implementation of pairing-based cryptography. Cryptology ePrint Archive, Paper 2009/556, 2009. https://eprint.iacr.org/2009/556.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. Cryptology ePrint Archive, Paper 2016/116, 2016. https://eprint.iacr.org/2016/116.

[cha] https://hackmd.io/@axiom/SJw3p-qX3.

[Cir] circom-pairing. https://github.com/yi-sun/circom-pairing. Accessed: 2024-04-03.

[CL69] S. Chowla and H. London. *Bounds on the n-th power residues (mod p)*, pages 679–680. Canad. Math. Bull. 12, 1969.

[DD95] Joseph B. Dence and Thomas P. Dence. Cubic and quartic residues modulo a prime. *Missouri Journal of Mathematical Sciences*, 7:24–31, 1995.

[Eag22] Liam Eagen. Zero knowledge proofs of elliptic curve inner products from principal divisors and weil reciprocity. Cryptology ePrint Archive, Paper 2022/596, 2022. `https://eprint.iacr.org/2022/596`.

[ext] `https://www.math.uchicago.edu/~may/VIGRE/VIGRE2009/REUPapers/Moy.pdf`.

[gar] `https://github.com/keep-starknet-strange/garaga/`.

[GG87] Frank Gerrish and David J. Garling. A course in galois theory. 1987.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. `https://eprint.iacr.org/2016/260`.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

[HH39] G. H. Hardy and By G. H. Hardy. An introduction to the theory of numbers. *The Mathematical Gazette*, 23:174, 1939.

[Hou22] Youssef El Housni. Pairings in rank-1 constraint systems. Cryptology ePrint Archive, Paper 2022/1162, 2022. `https://eprint.iacr.org/2022/1162`.

[HSV06] F. Hess, N. P. Smart, and F. Vercauteren. The eta pairing revisited. Cryptology ePrint Archive, Paper 2006/110, 2006. `https://eprint.iacr.org/2006/110`.

[IR90] K. Ireland and M.I. Rosen. *A Classical Introduction to Modern Number Theory*. Graduate Texts in Mathematics. Springer, 1990.

[KO62] Anatolii Karatsuba and Yu Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595, 12 1962.

[KST21] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Paper 2021/370, 2021. `https://eprint.iacr.org/2021/370`.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Polyno mial co m mit ments. 2010.

[Mil02] Victor Miller. Short programs for functions on curves. 09 2002.

[Mil04] Victor S. Miller. The weil pairing, and its efficient calculation. *J. Cryptol.*, 17(4):235–261, sep 2004.

[SBC+08] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. Cryptology ePrint Archive, Paper 2008/490, 2008. `https://eprint.iacr.org/2008/490`.

[Sco19] Michael Scott. Pairing implementation revisited. Cryptology ePrint Archive, Paper 2019/077, 2019. https://eprint.iacr.org/2019/077.

[Sil09] J.H. Silverman. *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics. Springer New York, 2009.

[Suc] Telepathy. https://github.com/succinctlabs/telepathy-circuits. Accessed: 2024-04-03.

[Ton91] Alberto Tonelli. Bemerkung über die auflösung quadratischer congruenzen. *Nachrichten von der Königl. Gesellschaft der Wissenschaften und der Georg-Augusts-Universität zu Göttingen*, 1891:344–346, 1891.

[Ver08] F. Vercauteren. Optimal pairings. Cryptology ePrint Archive, Paper 2008/096, 2008. https://eprint.iacr.org/2008/096.

[Was03] Lawrence C. Washington. Elliptic curves: Number theory and cryptography. 2003.