Constraint-Friendly Map-to-Elliptic-Curve-Group Relations and Their Applications

Jens Groth[1], Harjasleen Malvai[2], Andrew Miller[3], and Yi-Nuo Zhang[4]

[1] Nexus, jens@nexus.xyz
[2] UIUC and IC3, hmalvai2@illinois.edu
[3] Teleport, UIUC and IC3, soc1024@illinois.edu
[4] UC Berkeley, yinuo.yz@gmail.com

Abstract. Hashing to elliptic curve groups is a fundamental operation used in many cryptographic applications, including multiset hashing and BLS signatures. With the recent rise of zero-knowledge applications, they are increasingly used in constraint programming settings. For example, multiset hashing enables memory consistency checks in zkVMs, while BLS signatures are used in proof of stake protocols. In such cases, it becomes critical for hash-to-elliptic-curve-group constructions to be constraint-friendly such that one can efficiently generate succinct proofs of correctness. However, existing constructions rely on cryptographic hash functions that are expensive to represent in arithmetic constraint systems, resulting in high proving costs.
We propose a constraint-efficient alternative: a *map-to-elliptic-curve-group relation* that bypasses the need for cryptographic hash functions and can serve as a drop-in replacement for hash-to-curve constructions in practical settings, including the aforementioned applications. Our relation naturally supports non-deterministic map-to-curve choices making them more efficient in constraint programming frameworks and enabling efficient integration into zero-knowledge proofs. We formally analyze the security of our approach in the elliptic curve generic group model (EC-GGM).
Our implementation in `Noir/Barretenberg` demonstrates the efficiency of our construction in constraint programming: it achieves over $23\times$ fewer constraints than the best hash-to-elliptic-curve-group alternatives, and, enables 50–$100\times$ faster proving times at scale.

Keywords: Zero-knowledge proofs, elliptic curve cryptography, constraint programming, zkVM, generic group model, multiset hashing

1 Introduction

Hash-to-group functions map arbitrary messages to pseudorandom group elements and are important building blocks in cryptographic protocols based on cyclic groups. Two canonical applications are multiset hashing and BLS signatures.

Multiset hash functions extend collision-resistant hashing to unordered collections. They produce succinct digests while also supporting efficient *incremental updates*: given the digests of two multisets, one can compute the digest of their union via a simple algebraic operation on the digests. A widely adopted construction, introduced by Clarke et al. [CDvD$^+$03], builds multiset hashes from a hash-to-group function $\mathsf{HashToGroup} : \mathcal{M} \to G$, where $\mathcal{M}$ is the message space and G is a cryptographic group. The digest for a multiset S is computed as the group sum:

$$\mathsf{Digest}(S) = \sum_{m \in S} \mathsf{HashToGroup}(m).$$

Another prominent application of hash-to-group functions is BLS signatures [BLS04]. To sign a message m, the signer first hashes it to a group element $h_m = \mathsf{HashToGroup}(m)$, then computes the signature as $\sigma = h_m^{\mathsf{sk}}$, where sk is the signing key.

Constraint Programming and Zero-Knowledge Applications. Recent advances in zero-knowledge (ZK) proof systems have popularized the *constraint programming* model, where computations are encoded as sets of algebraic constraints. In this paradigm, rather than detailing computation steps that output a result, one specifies a set of constraints that the result, along with a set of auxiliary variables (called *witnesses*) must satisfy in order to be correctly computed. The result is considered valid if some *witness* assignment to these variables satisfies each constraint. This declarative programming approach is used by many modern zero-knowledge proof systems.

As constraint programming becomes central to zero-knowledge applications, many cryptographic primitives are being adapted to this framework. A concrete example is zero-knowledge virtual machines (zkVMs) such as RISC Zero [RIS25], SP1 [Lab25], Jolt [AST24] and Nexus [ACG+25], where the virtual machine must generate proofs attesting to correct execution of a program in the VM. As part of this proof, they must demonstrate memory read/write consistency across the entire execution trace. A common approach is *offline memory checking* [BEG+91], wherein the zkVM emits read and write logs as multisets and enforces constraints to ensure: (1) each memory access is properly recorded, and (2) the final read and write logs are equal as multisets. Crucially, as the logs grow large over time, maintaining them explicitly becomes costly. To address this, multiset hash functions can be used to compress each log into a succinct rolling digest, enabling efficient equality checks [SAGL18,Roy25]. Consequently, multiset hashing must itself be expressible within the constraint system.

Similarly, in zero-knowledge Proof-of-Stake (zkPoS) protocols, the prover must demonstrate knowledge of many valid BLS signatures over a sequence of blocks. This requires programming BLS verification into a constraint system [Hyp23].

Importantly, in these applications, the proof generation time scales with the number of constraints, so it is important for cryptographic primitives to be *constraint-friendly*i.e., expressible using a minimal number of constraints with low overhead.

Hash-to-Group is not Constraint-Friendly. Given the importance of hash-to-group functions as a building block of the aforementioned primitives, it is natural to ask whether they can be made *constraint-friendly*. Unfortunately, standard constructions are poorly suited for constraint programming environments. These constructions typically combine an *inner cryptographic hash function* (e.g., SHA-256) with an *outer map-to-group* function, which interprets the hash output as a group element.

The inner hash function is especially problematic in this setting. General-purpose hashes such as SHA-256 induce *thousands of constraints* when compiled into arithmetic circuits, significantly inflating proof generation costs. Constraint-friendly alternatives like MiMC [AGR+16] or Poseidon [GKR+21] reduce this burden and offer better efficiency, but even these hash functions typically require *hundreds of constraints* per invocation. This overhead becomes prohibitive when applied at scale, as in zkVM memory checks (which may involve hundreds of thousands of invocations) or in batch BLS signature verification in zkPoS.

The inefficiency of such hash-to-group constructions motivates the question:

Can we design a hash-to-group replacement that avoids cryptographic hash functions altogether, while preserving meaningful security and enabling constraint-friendly implementation in practical applications?

1.1 Our Contributions

In this paper, we answer the above question in the affirmative by developing a constraint-friendly alternative grounded in the generic group model.

- **No Reliance on Cryptographic Hash Functions:** We show that with some restrictions on the message space, in the Generic Group Model (GGM), a map-to-group function alone can achieve the same security guarantees as any hash-to-group function. In the basic version we restrict the message sizes but we still capture many practical applications, such as multiset hashing in zkVMs and BLS signatures.
- **Defining Map-to-Group Relations:** We then introduce the notion of a *map-to-group relation* as a constraint-programming-compatible counterpart to standard map-to-group functions. Unlike deterministic functions, a relation formulation allows for nondeterministic input (witnesses), which enables flexible, low-overhead instantiation in constraint systems. We specifically focus on map-to-elliptic-curve-group relations, since elliptic curves groups are widely used in practice due to their efficiency.
- **Applications to Constraint Programming:** We apply our map-to-elliptic-curve-group relation to efficiently instantiate multiset hash and BLS signature in constraint systems. In both cases, we reformulate the primitives as relations to align with the constraint programming paradigm, and we formally prove their security under the elliptic curve generic group model.
- **Implementation and Benchmarking:** We implement our map-to-elliptic-curve-group relation and compare it against standard hash-to-elliptic-curve-group constructions. Our evaluation shows that a single invocation of our relation can be expressed in just 30 constraints. In contrast, hash-to-elliptic-curve-group constructions incur significantly higher costs depending on the hash function used, ranging from approximately 350 constraints (using MiMC) to over 7,000 constraints (using SHA-256).

These constraint savings translate directly into improved proving performance. For example, proving 2^{15} instances of our map-to-curve relation takes approximately 7.6 seconds, while even the most efficient hash-based baseline requires over 400 seconds.

1.2 Technical Overview

Our approach begins with the observation that, in the *Generic Group Model (GGM)* a group element encodes a random field element, i.e., it has a random discrete logarithm. The encoding therefore plays a role analogous to a cryptographic hash modeled as a random oracle. Specifically, in a cyclic group of prime order p, once a generator is fixed, every group element can be uniquely identified by its discrete logarithm $u \in \mathbb{F}_p$. From this viewpoint, we will treat group elements interchangeably with their corresponding discrete logs. In the GGM, these discrete logs are then abstractly "garbled" via a uniformly random bijection:

$$\mathsf{enc} : \mathbb{F}_p \to \texttt{Encodings},$$

which randomly assigns each discrete log $u \in \mathbb{F}_p$ to a unique encoding $\mathsf{enc}(u)$. Without direct access to the underlying discrete logs, this encoding function obscures the algebraic structure of the group from the adversary.

Viewed through this lens, the standard construction of a hash-to-group function can be understood as a composition of three layers:

- an *inner cryptographic hash* that in the random oracle model maps some given message space $\mathcal{M}$ to uniformly random hash values;
- a *middle map-to-group* which maps the hash values to elements in encoding domain `Encodings`;
- an *outer encoding function* (e.g., enc) that maps these encodings to uniformly random discrete logarithms in $\mathbb{F}_p$.

However, if the primary objective is simply to map messages to uniformly random discrete logs in $\mathbb{F}_p$, without requiring that group encodings themselves appear random, then the inner cryptographic hash function becomes redundant particularly when the message space $\mathcal{M}$ can be directly mapped to `Encodings` via a map-to-group function. A similar insight appears heuristically in Browns work [Bro08], which conjectures bypassing the inner hash entirely in building Encrypted Elliptic Curve Hashes (EECH). We extend this observation to a broader setting, observing that the security of many practical applicationssuch as aforementioned BLS signatures and multiset hashrequire only that messages be mapped to uniformly random discrete logarithms.

Addressing collisions via restricting the message space $\mathcal{M}$. We thus explore the possibility of replacing a hash-to-group function with just the simpler *map-to-group function*, where security relies on GGM. However, to securely make this replacement, we must ensure that the mapping from the message space $\mathcal{M}$ to discrete logarithms remains *random*even in the presence of adversarial queries to the GGM oracle. Whereas for hash-to-group this property is inherently guaranteed in the random oracle model, preserving it in the map-to-group setting under the GGM requires more careful design and analysis.

In the GGM, adversaries interact with a group operation oracle that takes any adversary's input encodings $\mathsf{enc}(u)$ and $\mathsf{enc}(v)$ and returns encoding $\mathsf{enc}(u + v)$. We say a collision occurs for this query if there exists a message $m \in \mathcal{M}$ which is mapped to the encoding $\mathsf{enc}(u + v)$. A collision breaks the aforementioned guarantee since the adversary may correlate the discrete log of m with a known relation.

To mitigate this risk, we observe that restricting the message space $\mathcal{M}$ can significantly reduce the probability of collision. Since the encoding function enc in the GGM is modeled as a uniformly random bijection from discrete logs $\mathbb{F}_p$ to encodings `Encodings`, as long as the map-to-group function is independent of enc, the probability that an oracle output (e.g. $\mathsf{enc}(u + v)$) falls within the image of the map-to-group is approximately $|\mathcal{M}|/p$. Thus, for an adversary making q oracle queries, the overall probability of any such collision is bounded by:

$$\Pr[\text{collision}] \leq \frac{q \cdot |\mathcal{M}|}{p}.$$

As a concrete example, in a group of order $p = 2^{256}$, setting $|\mathcal{M}| = 2^{128}$ gives 128 bits of collision resistance.

Map-to-elliptic-curve-group relation. Driven by real applications, from this point we focus on elliptic curve groups for their practical efficiency. Informally, an elliptic curve over a finite field $\mathbb{F}_q$ is defined by a curve equation over (x, y), where the set of solutions $(x, y) \in \mathbb{F}_q^2$, together with a special point at infinity forms a group $E(\mathbb{F}_q)$. Cryptographic applications typically work with a prime-order cyclic subgroup $G \subseteq E(\mathbb{F}_q)$. Let p be the order of G, then the value h such that $|E(\mathbb{F}_q)| = h \cdot p$ is called the *cofactor*.

In the Generic Group Model for elliptic curves proposed by [GS22], each point $(x, y) \in G$ is treated as an abstract encoding of a unique discrete logarithm $u \in \mathbb{F}_p$. To remain compatible with this abstraction, we seek map-to-group constructions that do not rely on specific properties of the elliptic curve equation.

A natural candidate is the classical *increment-and-check* method, which makes no structural assumptions about the curve and is therefore well-suited for modeling generic elliptic curve groups. Informally, given a message $m \in \mathcal{M}$, the method interprets m as a candidate x-coordinate, checks whether there exists a corresponding y such that $(x = m, y)$ lies on the curve and encodes a valid group element, and if not, iteratively increments the input $(x = m + 1, m + 2, \ldots)$ until such an encoding is found.

While conceptually simple, this method introduces two practical challenges. First, in traditional implementations, the number of iterations varies with the input, which can both incur a cost and potentially leak information. In constraint programming, however, this issue is mitigated by treating the tweak value (i.e., the number of increments) as part of the *witness*, effectively rendering the construction *deterministic and constant-time* from the verifiers perspective. This motivates our formalization of a **map-to-elliptic-curve-group relation**, where non-determinism is captured by witness variables.

Second, the efficiency of the increment-and-check method depends on the cofactor h: each trial succeeds with probability approximately $1/(2h)$. Fortunately, as shown in Table 1, many widely used elliptic curves have small cofactors, making the method efficient in practice.

In our applications, it is important for the map-to-elliptic-curve-group relation to satisfy *injectivity* and *inverse exclusion* in order to prevent collisions and trivial attacks. For example, if the relation is not injective, then in BLS signatures, two distinct messages can map to the same group element, enabling trivial forgeries. We define those properties formally in section 3.3.

Applications in constraint programming. We demonstrate the utility of our map-to-elliptic-curve-group relation for two core constraint programming applications:

- **Multiset Hash.** Following the blueprint of [CDvD$^+$03], we construct a constraint-friendly multiset hash from an injective and inverse-excluding map-to-elliptic-curve-group relation. This enables efficient enforcement of memory consistency in *zero-knowledge virtual machines (zkVMs)*.
- **BLS Signature Verification.** We construct a BLS signature scheme with constraint-friendly verification by replacing its hash-to-elliptic-curve group function with an injective map-to-elliptic-curve-group relation. This is particularly useful in *zero-knowledge Proof of Stake (zkPoS)* protocols, where succinct proofs are used to attest to the validity of aggregate BLS signatures over many blocks.

1.3 Related Work

Map-to-elliptic-curve-group. The main primitive which we introduce in this work is a constraint-friendly map-to-elliptic-curve group relation. There is already a body of literature on (non-)deterministic *map-to-elliptic-curve* functions, described below, which map input values to curve points of elliptic curves. Those curve points can then be mapped to elements in the elliptic curve group via a procedure called "clearing cofactor", as described in Section 4.1. It's worth noting that all those functions are designed to be fast to execute, instead of being constraint-friendly.

Non-deterministic Approach. The *increment-and-check* method, described by Boneh et al. [BLS04], is a Las Vegas type probabilistic algorithm. It works by viewing the message as an integer and increments until a valid point on the curve is found. More details can be found in section 4.1. This method is concretely efficient in practice and works for any elliptic curve.

Deterministic Approach. Due to the number of iterations varying with the input, the *increment-and-check* method can leak non-trivial input information. Thus a line of works aim at providing *deterministic* mappings with constant runtime for elliptic curve groups. Icart [Ica09] proposed a different deterministic approach which is efficient for ordinary elliptic curves over fields with characteristic $p \equiv 2 \mod 3$. Farashahi et al. [FSV09] and Fouque

and Tibouchi [FT10] further analyzed and generalized these results to Hessian curves and certain hyperelliptic curves. Besides, to enable unique message recovery, Fouque et al. [FJT13] also considered injective mappings to elliptic curves. Shallue and van de Woestijne [SvdW06] generalized and simplified previous constructions by Skaba [Ska05] to provide deterministic mappings to any elliptic curves. However, their method does not work for all field characteristics. Ulas [Ula07] further simplified the construction, but restricted its application to Weierstrass-form elliptic curves, excluding curves with j-invariant 0 or 1728. This exclusion is significant because many pairing-friendly curves, such as Barreto-Naehrig (BN) and Barreto-Lynn-Scott (BLS) curves, have j-invariants of either 0 or 1728, limiting the applicability of the Ulas method for cryptographic protocols that rely on these curves. To remedy this issue, Fasano et al. [WB19] present efficient and constant-time mapping for the BLS12-381 elliptic curve, extending the applicability of the [Ula07] map to $|\mathbb{F}| \equiv 1 \mod 4$ and even-order extension fields, optimizing these mappings for performance and implementation.

More recently, Koshelev proposed new constructions tailored for highly 2-adic fields. In his first work [Kos23], the construction is limited to elliptic curves admitting an $\mathbb{F}_q$-isogeny of degree 3, achieving $O(\log q)$ complexity but with restricted applicability. This was subsequently improved in follow-up work [Kos24], which removes the isogeny constraint, covers all elliptic curves with $j \neq 0, 1728$, and leverages a constant-time variant of the CipollaLehmerMller square-root algorithm to obtain both broader applicability and better performance in highly 2-adic settings.

Hash-to-elliptic-curve-group. As discussed previously, standard hash-to-elliptic-curve-group constructions typically combine an *inner cryptographic hash function* with an *outer map-to-elliptic-curve-group* function, which interprets the hashed output as a group element on an elliptic curve. As a concrete example, Spice Hash [SAGL18] builds a hash-to-elliptic-curve-group by composing an inner hash function (instantiated using the MiMC hash function) with a map-to-group based on Elligator-2 construction [BHKL13].

Non-deterministic Approach. This approach is hinted by the aforementioned *increment-and-check* method. However, for security reasons, the increment procedure must be performed before the hash is taken. That is, we iteratively hash increments of the message until the result corresponds to a valid point on the curve. This method is the most efficient and widely used in practice. For this reason, in our evaluation, we adopt this method as a baseline for comparison. It's worth pointing out that its non-determinism can be naturally handled in constraint programming via witness variables, and does not pose security concerns for zero-knowledge applications.

Deterministic Approach. In contrast, deterministic constructions introduce further complexity. A core challenge is that deterministic map-to-elliptic-curve-group functions often fail to cover the entire curve uniformly. For instance, the Icart method [Ica09] deterministically maps field elements to points on an elliptic curve, covering more than $5/8$ of the curve but lacking injectivityeach point can have up to four preimages. Consequently, even when composed with a random oracle, the resulting distribution over group elements deviates from uniformity, undermining the security properties expected from an ideal hash-to-group.

To overcome this, a line of work has sought to construct hash-to-elliptic-curve-group functions that are *indifferentiable from a random oracle*. This property ensures that the hash behaves as a random function from the adversarys perspective.

Brier et al. [BCI$^+$10] introduced the first general indifferentiability framework for hash-to-curve functions. Their construction relies on two independent random oracles: the first maps the input to a field element which is then passed through a map-to-curve function (e.g., Icart or Simplified SWU), while the second oracle maps the input directly to a point. The outputs are then combined via group addition. This ensures that no single deterministic mapping dominates the output distribution.

Farashahi et al. [FFS$^+$13] extended this approach, showing that many deterministic map-to-curve functions can be securely composed with such randomized techniques to yield indifferentiable hash-to-curve constructions. Kim and Tibouchi [TK17] further refined this framework by offering tunable parameters that trade off performance against tightness of security guarantees, resulting in the widely adopted Simplified SWU method.

Most recently, Chvez-Saab, Rodrguez-Henrquez, and Tibouchi introduced *SwiftEC* [CSRHT22], which resolves the long-standing open problem of achieving indifferentiable hashing to (almost all) elliptic curves at the cost of a *single* field exponentiation. Building on this, Koshelev [Kos26] further generalized the SwiftEC paradigm, giving compact universal formulas valid for all elliptic curves with $3 \mid q - 1$, thereby unifying prior approaches (including SwiftEC) into a single, simpler framework. Together, these results represent the current state of the art in deterministic, indifferentiable hashing to elliptic curves.

We refer the interested reader to RFC 9380 [FHSS+23] for a comprehensive survey of knowledge in this area, capturing most previous research on mapping and hashing to elliptic curves, formal definitions, and standardized protocols for elliptic curve hashing.

As stated earlier, all these constructions rely on cryptographic hash functions as a preprocessing step though, which is costly when expressed in constraint systems. Our work bypasses this inner hash layer by leveraging the encoding abstraction offered by the Generic Group Model, which offers a slightly weaker yet still meaning security guarantee for our targeted applications.

2 Preliminaries

Notation: We denote the security parameter by λ, and let $\mathsf{negl}(\lambda)$ represent a negligible function. The notation $\{0,1\}^*$ refers to bit strings of arbitrary length, while $\{0,1\}^k$ is the set of all k-bit strings. We denote by G a cryptographic group with prime order $p = O(2^\lambda)$. Since the group is cyclic, any group element can be identified with its unique discrete logarithm $u \in \mathbb{F}_p$ with respect to some fixed group generator. An algorithm is said to be efficient if it runs in probabilistic polynomial time (PPT) with respect to the security parameter λ. The bit-security level refers to the number of bits of computational effort required to break a cryptographic scheme. For instance, 128-bit security means that an adversary would need to perform 2^{128} operations on average to break the scheme.

2.1 Constraint Programming

Traditional programming typically involves providing explicit step-by-step instructions to solve a problem, using control structures such as loops and conditionals to direct the flow of computation. In contrast, *constraint programming* defines a problem by specifying a set of constraints that must be satisfied by the variables. Rather than describing how to compute a solution, one describes what properties a valid solution must have. A solver is then used to find a satisfying assignment, often aided by auxiliary values called a *witness*. This separates the problem specification from the computation process. A collection of widely used constraint programming paradigms express all constraints as algebraic relations over finite fields:

Definition 1 (Constraint Satisfaction over Finite Fields). Constraint Programming (CP) *over a finite field* $\mathbb{F}$ *refers to solving constraint satisfaction problems (CSPs) where all variables range over* $\mathbb{F}$*. A CSP is a tuple* $(X_{\textit{in/out}}, X_{\textit{wit}}, \mathcal{C})$*, where:*

- $X_{\textit{in/out}} = \{x_1, x_2, \ldots, x_k\}$ *is the set of* input and output variables.
- $X_{\textit{wit}} = \{w_1, w_2, \ldots, w_\ell\}$ *is the set of* witness variables.
- $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ *is a set of algebraic constraints over the field* $\mathbb{F}$*, each defined over a subset of variables from* $X_{\textit{in/out}} \cup X_{\textit{wit}}$*.*

A solution *to the CSP is an assignment of values in* $\mathbb{F}$ *to the witness variables such that all constraints in* $\mathcal{C}$ *are satisfied under the fixed input/output values.*

Example 1 (Zero-Knowledge Circuits). A key application of constraint programming is the construction of zero-knowledge (ZK) circuits, where a prover demonstrates knowledge of a witness that satisfies a given constraint system, without revealing the witness itself. Modern proof systems-such as those built using the `circom` compiler-specify these circuits as sets of arithmetic constraints over a finite field $\mathbb{F}$. These constraints are then compiled into a *Rank-1 Constraint System (R1CS)*, a canonical format in which each constraint takes the form $(a \cdot x)(b \cdot x) = c \cdot x$ for fixed vectors $a, b, c \in \mathbb{F}^n$ and variable vector $x \in \mathbb{F}^n$.

It is worth noting that due to algebraic restrictions imposed by proof systems, the choice of field $\mathbb{F}$ is limited in practice. Common choices include the scalar field $\mathbb{F}_q$ of a pairing-friendly elliptic curve (as defined in definition 2), or an FFT-friendly prime field such as the *Goldilocks field* $\mathbb{F}_{2^{64}-2^{32}+1}$. These fields enable efficient arithmetic and are widely supported in ZK backends like Halo2 and Plonky3.

2.2 Elliptic Curve Group

Definition 2 (Elliptic Curve). *Let* $\mathbb{F}_q$ *be a finite field of prime order* $q > 3$*. A typical* elliptic curve E *over* $\mathbb{F}_q$ *is defined by some curve equation over* (x, y)*. For example, a Weierstrass equation has the form:*

$$y^2 = f(x), \text{ where } f(x) = x^3 + ax + b.$$

The set of points $E(\mathbb{F}_q)$ consists of all pairs $(x, y) \in \mathbb{F}_q^2$ satisfying the equation above, together with a distinguished point at infinity $\mathcal{O}$, form a finite Abelian group with $\mathcal{O}$ as the identity element. In particular, due to the underlying geometry, the inverse of a point (x, y) is $(x, -y)$.

Let $n = |E(\mathbb{F}_q)|$ denote the order of the curve. By Hasse's theorem, we know:

$$|n - (q + 1)| \leq 2\sqrt{q}.$$

In cryptographic applications, we typically focus on a cyclic subgroup $G \subseteq E(\mathbb{F}_q)$ of prime order p. The order of the entire curve group satisfies $n = h \cdot p$, where h is the *cofactor*. The cofactor varies significantly among elliptic curves commonly used in cryptography, as summarized in Example 2.

Example 2 (Cofactors of Commonly Used Elliptic Curves). Table 1 provides the cofactors h for several elliptic curves in Weierstrass form that are frequently used in cryptographic applications:

Curve Name	**Cofactor** h
secp256k1	1
secp256r1 (P-256)	1
BN254 (G_1)	1
Curve25519	8
JubJub	8
BLS12-381 (G_1)	$\approx 2^{128}$

Table 1: Cofactors of commonly used elliptic curves.

We observe that many widely adopted elliptic curves admit small cofactors. This is advantageous because most curve points lie directly within the subgroup, simplifying subgroup membership checks and enabling compact group-element representations.

However, curves supporting pairing operations, such as BLS12-381 and BN254, often exhibit large cofactors in their **secondary** groups (e.g., G_2).

2.3 Elliptic Curve Generic Group Model (EC-GGM)

A cryptographic group is a prime order group where given any generator of the group, every group element can be identified with a unique discrete logarithm with respect to the generator that defines the algebraic structure of the group. The *Generic Group Model (GGM)*, introduced by Shoup [Sho97], provides a heuristic framework for analyzing generic adversaries interacting with a cryptographic group by restricting adversaries to solely relying on abstract interfaces of the group. Specifically, in the GGM, adversaries have no direct access to the underlying group elements (discrete logarithms); instead, they can only observe some encodings of group elements. Notably, the relation between those encodings and the actual group elements (discrete logs) will be entirely garbled by a uniformly randomly sampled bijection unknown to the adversary. The adversary can then perform group operations through abstract interfaces, which takes those encodings as input, perform the operation on the underlying discrete logs, and return the encoding of the resulting group elements. Security proofs in the GGM demonstrate resistance to attacks exploiting only these generic operations, serving as necessary (though not always sufficient) conditions for practical security.

The Elliptic Curve Generic Group Model (EC-GGM), introduced by [GS22], refines the Generic Group Model (GGM) to account for the algebraic structure of elliptic curves. As defined in definition 2, points on an elliptic curve are of the form $(x, y) \in \mathbb{F}_q^2$, with a distinguished identity element $\mathcal{O}$. The group law exhibits a natural symmetry: each point (x, y) has an inverse $(x, -y)$ satisfying $(x, y) + (x, -y) = \mathcal{O}$. The EC-GGM explicitly models this symmetry, ensuring that inversion behavior is preserved in the abstraction.

Definition 3 (Elliptic Curve Generic Group Model (EC-GGM)). *Let $G \subseteq E(\mathbb{F}_q)$ be a cyclic elliptic curve subgroup of prime order p, generated by a point $\mathcal{G} = (x_1, y_1)$. In the EC-GGM, each group element is identified by its discrete logarithm $u \in \mathbb{F}_p$ and represented via a* ***randomly sampled bijection***

$$\mathsf{enc} : \mathbb{F}_p \to G \subseteq E(\mathbb{F}_q), \quad u \mapsto (x_u, y_u),$$

which satisfies the following properties:

$$\mathsf{enc}(0) = \mathcal{O}, \quad \mathsf{enc}(1) = (x_1, y_1), \quad \textit{and if } \mathsf{enc}(u) = (x_u, y_u)$$
$$\textit{then } \mathsf{enc}(-u) = (x_u, y_{-u}) = (x_u, -y_u) \quad \forall u \in \mathbb{F}_p.$$

Adversaries are only permitted to interact with group elements through abstract interfaces that operate on these encodings, thereby concealing the underlying discrete logarithmic structure. The interface includes:

Addition Oracle. *An oracle* add_G *that simulates group addition: given* $(x_u, y_u) = \mathsf{enc}(u)$ *and* $(x_v, y_v) = \mathsf{enc}(v)$*, the oracle returns:*

$$\mathsf{add}_G((x_u, y_u), (x_v, y_v)) := \mathsf{enc}(u + v) = (x_{u+v}, y_{u+v}).$$

EC-GGM Encoding Over Inversion Classes: We partition the space of group elements into equivalence classes under inversion. This abstraction allows us to factor out the effect of elliptic curve symmetry $(x, y) \approx (x, -y)$, yielding a cleaner characterization of the EC-GGM encoding function.

Definition 4 (Inversion Classes over $\mathbb{F}_p$ and G). *Let $G \subseteq E(\mathbb{F}_q)$ be a cyclic subgroup of prime order p, and let $\mathsf{enc} : \mathbb{F}_p \to G$ be the EC-GGM encoding function.*

- *Define an equivalence relation $\sim$ on $\mathbb{F}_p$ by $u \sim v$ if $u = \pm v$. The set of equivalence classes is denoted $\mathbb{F}_p^{\pm} := \mathbb{F}_p/\{\pm 1\}$, with representatives $[u] := \{u, -u\}$.*
- *Define an equivalence relation $\approx$ on G by $(x_u, y_u) \approx (x_v, y_v)$ if $(x_u, y_u) = (x_v, \pm y_v)$. The corresponding set of inversion classes is denoted $G^{\pm} := G/\{\pm 1\}$, with elements written as $[(x_u, y_u)] := \{(x_u, y_u), (x_u, -y_u)\}$.*

Now, enc is a random bijection satisfying $\mathsf{enc}(-u) = (x_u, -y_u)$ whenever $\mathsf{enc}(u) = (x_u, y_u)$. Therefore, enc is a random bijection which maps the inversion class $[u] \in \mathbb{F}_p^{\pm}$ to the group class $[(x_u, y_u)] \in G^{\pm}$, making $\overline{\mathsf{enc}}([u]) := [(x_u, y_u)]$ well defined.

Proposition 1 (Induced-Encoding Over Inversion Classes). *Let $\mathsf{enc} : \mathbb{F}_p \to G$ be the EC-GGM encoding function. Then enc induces a random bijection*

$$\overline{\mathsf{enc}} : \mathbb{F}_p^{\pm} \to G^{\pm}, \quad \overline{\mathsf{enc}}([u]) := [(x_u, y_u)].$$

Both sets have $\frac{p+1}{2}$ elements since 0 maps to $\mathcal{O}$, which is its own inverse.

3 Map-to-Elliptic Curve Group Relations

In the constraint programming setting, it is often more natural to express verification problems as relations rather than deterministic functions. Instead of requiring that an output be uniquely determined by the input, a relation both allows for nondeterministic advice and removes uniqueness, providing more flexibility when associating outputs to inputs.

Specifically, consider mapping some message space to a subgroup of an elliptic curve. Rather than treating the mapping as a deterministic function, we describe it by a relation:

Definition 5 (Map-to-Elliptic-Curve Group Relation). *Let $\mathcal{M}$ be a message space, $G \subseteq E(\mathbb{F}_q)$ a prime-order elliptic curve subgroup modeled under the EC-GGM, and $\mathcal{W}$ a witness space. A* map-to-elliptic-curve-group relation *is a relation*

$$\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G \times \mathcal{W},$$

where each tuple $(m, (x_{u_m}, y_{u_m}), w_m) \in \mathcal{R}_{\mathsf{M2G}}$ consists of a message $m \in \mathcal{M}$, a group element $(x_{u_m}, y_{u_m}) \in G$ corresponding to some discrete logarithm $u_m \in \mathbb{F}_p$, and a nondeterministic witness $w_m \in \mathcal{W}$ certifying membership in $\mathcal{R}_{\mathsf{M2G}}$.

3.1 Characterization of Map-to-Elliptic-Curve-Group Relations in EC-GGM

We now identify some key structural properties of map-to-elliptic-curve-group relations in the EC-GGM.

Theorem 1 (Uniformly Random Distribution of Discrete Log Classes). *Let $\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G \times \mathcal{W}$ be a map-to-elliptic-curve-group relation that is **independent** of the EC-GGM encoding function $\mathsf{enc} : \mathbb{F}_p \to G$. For each message $m \in \mathcal{M}$, define its associated inversion class set:*

$$S_m^{\pm} := \left\{ [u_m] \in \mathbb{F}_p^{\pm} \;\middle|\; (m, (x_{u_m}, y_{u_m}), w_m) \in \mathcal{R}_{\mathsf{M2G}} \right\}.$$

Then, in the EC-GGM, the set $S_m^{\pm} \subseteq \mathbb{F}_p^{\pm}$ is distributed as a uniformly random subset of $\mathbb{F}_p^{\pm}$.

Proof. Fix $m \in \mathcal{M}$. Since $\mathcal{R}_{\mathsf{M2G}}$ is independent of the EC-GGM encoding function enc, the group elements (x_{u_m}, y_{u_m}) appearing in the relation for this fixed m are determined independently of the encoding function.

By proposition 1, the EC-GGM encoding enc induces a uniformly random bijection $\overline{\mathsf{enc}} : \mathbb{F}_p^{\pm} \to G^{\pm}$. This means that for each group element (x_{u_m}, y_{u_m}) in the relation, its corresponding discrete logarithm class $[u_m] \in \mathbb{F}_p^{\pm}$ is sampled uniformly independently at random.

As a result, the set $S_m^{\pm}$, which contains all inversion classes $[u_m]$, is uniformly distributed over all subsets of $\mathbb{F}_p^{\pm}$ of corresponding size.

3.2 Efficient Constructibility

For the sake of witness generation in constraint programming, we are often interested in map-to-elliptic-curve-group relations where it is possible to efficiently generate valid relation instances $(m, (x_{u_m}, y_{u_m}), w_m) \in \mathcal{R}_{\mathsf{M2G}}$ for a given message $m \in \mathcal{M}$. Since guaranteeing success for all inputs may be too strict in practice, we allow for a negligible failure probability.

Definition 6 (Efficient Constructibility). *Let $\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G \times \mathcal{W}$ be a map-to-elliptic-curve-group relation, where $G \subseteq E(\mathbb{F}_q)$ is a prime-order elliptic curve subgroup, and let λ be a security parameter.*

We say that $\mathcal{R}_{\mathsf{M2G}}$ is efficiently constructible *if there exists a deterministic polynomial-time algorithm*

$$\mathsf{Construct}_{\mathsf{M2G}} : \mathcal{M} \to G \times \mathcal{W} \cup \{\perp\}$$

such that:

- *For all $m \in \mathcal{M}$, the algorithm returns either $\perp$ or a pair $((x_{u_m}, y_{u_m}), w_m)$ such that $(m, (x_{u_m}, y_{u_m}), w_m) \in \mathcal{R}_{\mathsf{M2G}}$.*
- *For all $m \in \mathcal{M}$, the probability of failure (over the random sampling of $\mathcal{R}_{\mathsf{M2G}}$) is*

$$\Pr\left[\mathsf{Construct}_{\mathsf{M2G}}(m) = \perp\right] \leq \mathsf{negl}(\lambda).$$

3.3 Injectivity and Inverse Exclusion

We now formalize two structural properties of map-to-elliptic-curve-group relationsinjectivity and inverse exclusionwhich will be crucial in our applications.

Definition 7 (Injective Relation). *We say that $\mathcal{R}_{\mathsf{M2G}}$ is* injective *if for any pair of tuples*

$$(m_1, (x_{u_{m_1}}, y_{u_{m_1}}), w_{m_1}),\ (m_2, (x_{u_{m_2}}, y_{u_{m_2}}), w_{m_2}) \in \mathcal{R}_{\mathsf{M2G}}$$

with distinct $m_1 \neq m_2$, we also have $(x_{u_{m_1}}, y_{u_{m_1}}) \neq (x_{u_{m_2}}, y_{u_{m_2}})$.

Notice that if $\mathcal{R}_{\mathsf{M2G}}$ is *injective*, then for any two distinct messages $m_1 \neq m_2 \in \mathcal{M}$ we have $S_{m_1}^{\pm} \cap S_{m_2}^{\pm} = \emptyset$.

Definition 8 (Inverse-Excluding Relation). *We say that $\mathcal{R}_{\mathsf{M2G}}$ is* inverse excluding *if for all tuples $(m, (x_{u_m}, y_{u_m}), w_m), (m', (x_{u_m}$ $\mathcal{R}_{\mathsf{M2G}}$ (where we allow $m = m'$), it holds that:*

$$(x_{u_m}, y_{u_m}) \neq (x_{u_{m'}}, -y_{u_{m'}}).$$

Equivalently, no two group elements occurring in $\mathcal{R}_{\mathsf{M2G}}$ belong to the same inversion class in $G^{\pm}$ as defined in definition 4.

4 Efficiently Constructible Map-to-Elliptic-Curve-Group Relations

We now present a map-to-elliptic-curve-group relation that is efficiently constructible. As a starting point, let's first define what the constructor should look like. This construction is inspired by the classic increment-and-check paradigm of Boneh et al. [BLS04].

4.1 Increment-and-Check Constructor

Let $G \subseteq E(\mathbb{F}_q)$ be an elliptic curve group with cofactor h. The classical increment-and-check method maps a message $u \in \mathbb{F}_q$ deterministically to a point (x, y) on an elliptic curve $E(\mathbb{F}_q)$. The basic idea is to try successive "tweaks" of the input and check whether they produce a valid point on the curve. Then a procedure called "clearing cofactor" is applied, projecting this point as $h \cdot (x, y)$, which becomes an element in G.

In this paper, we slightly modify this approach so that instead of first mapping to points on an elliptic curve and then clearing the cofactor, we aim to *directly map to elements of an elliptic curve group G* at a small cost of trying a few more tweaks. Moreover, we make it deterministic by fixing an upper bound on its number of iterations. This modified procedure, which assumes the cofactor to be small, is described in fig. 1.

Increment-and-Check Constructor

Inputs:

- An elliptic curve $E(\mathbb{F}_q)$ of the form $y^2 = f(x)$.
- An elliptic curve group $G \subseteq E(\mathbb{F}_q)$.
- A message $u \in \mathbb{F}_q$.
- A tweak bound $T \in \mathbb{N}$.

Procedure:

1. Initialize tweak counter $k \leftarrow 0$.
2. While $k < T$:
   (a) Let $x \leftarrow u + k$.
   (b) Compute and check whether $f(x)$ is a quadratic residue in $\mathbb{F}_q$.
      - If so, compute $y \leftarrow \sqrt{f(x)}$ // deterministically pick one of the roots
      - If $(x, y) \in G$, then output (x, y).
   (c) Otherwise, increment $k \leftarrow k + 1$.
3. If no valid point is found after T iterations, declare failure.

Fig. 1: Deterministic increment-and-check constructor for map-to-elliptic-curve-group.

Failure Probability. We now show that the above constructor satisfies definition 6. To bound the failure probability, we rely on a well-known heuristic assumption underlying [BLS04]. Informally, we assume that all possible x-coordinates which correspond to points in G are randomly distributed within $\mathbb{F}_q$. We will call those x-coordinates "valid" coordinates.

Under this heuristic, the probability that any $x \in \mathbb{F}_q$ is valid is approximately $\frac{1}{2h}$, where h is the cofactor of G. (The factor $1/2$ accounts for the necessity that $f(x)$ must be a quadratic residue.) Thus, the probability that none of the tweaks in T succeeds in finding a valid coordinate is approximately $\left(1 - \frac{1}{2h}\right)^{|T|}$. In particular, for small constant cofactor h and $|T| = O(\lambda)$, the failure probability becomes negligible. It's worth noting that most commonly used elliptic curve groups admit small cofactors, as highlighted in example 2.

4.2 Map-to-Elliptic-Curve-Group Relation

We now define a map-to-elliptic-curve-group relation that can be efficiently constructed using the increment-and-check procedure described in fig. 1. This relation is defined over the message space $\mathcal{M}$ and elliptic curve group G, and includes a bounded tweak $k \in [0, T)$ as part of the witness. Notice that in this construction, for each $m \in \mathcal{M}$,

we have $|S_m^{\pm}| \leq T$, where $S_m^{\pm}$ is the set of discrete log inversion classes associated with message m, as previously defined in theorem 1.

In many applications, it is desirable for the relation to satisfy the *inverse-exclusion* property, meaning that for any $x \in \mathbb{F}_q$, only one of $(x, y) \in G$ or $(x, -y) \in G$ will be included in the relation. To enforce inverse-exclusion in such cases, we simply require a canonical method to select a representative from the pair $\{y, -y\}$. To this end, notice that whenever $q \equiv 3 \pmod 4$, at most one of $\pm y$ is a quadratic residue, which we will select as the unique representative. This condition can be efficiently enforced in constraint programming by requiring the constraint $y = z^2$ for some additional witness $z \in \mathbb{F}_q$.

Map-to-Elliptic-Curve-Group Relation

Parameters:

- A prime-order subgroup $G \subseteq E(\mathbb{F}_q)$ of an elliptic curve $E(\mathbb{F}_q)$, where q is a large prime with $q \equiv 3 \pmod 4$.
- A message space $M \subseteq \mathbb{F}_q$ and tweak space $[0, T)$ such that $M \cdot T < q$.

Relation $\mathcal{R}_{\mathsf{M2G}} \subseteq M \times G \times \mathcal{W}$:

- A triple $(m, (x, y), w)$ where $w = (k, z)$ belongs to $\mathcal{R}_{\mathsf{M2G}}$ if:
  1. $m \in M, k \in [0, T)$,
  2. $x := m \cdot T + k$,
  3. $y = z^2$,
  4. $(x, y) \in G$.

Fig. 2: An efficiently constructible map-to-elliptic-curve-group relation with canonical y-selection via quadratic residuosity.

Claim. The map-to-elliptic-curve-group relation $\mathcal{R}_{\mathsf{M2G}}$ defined in fig. 2 is both injective and inverse excluding.

Proof. First, we show that $\mathcal{R}_{\mathsf{M2G}}$ is inverse excluding. By construction, the relation only includes points (x, y) such that y is the quadratic residue. Since exactly one of $\pm y$ can be a quadratic residue when $q \equiv 3 \pmod 4$, the relation must be inverse excluding.

Next, we show that $\mathcal{R}_{\mathsf{M2G}}$ is injective. Suppose there exist two distinct messages $m_1 \neq m_2$ and corresponding valid triples $(m_1, (x_1, y_1), w_1)$, $(m_2, (x_2, y_2), w_2) \in \mathcal{R}_{\mathsf{M2G}}$ such that $x_1 = x_2$. Let $w_1 = (k_1, z_1)$ and $w_2 = (k_2, z_2)$. By construction, we have:

$$x_1 = m_1 \cdot T + k_1, \quad x_2 = m_2 \cdot T + k_2.$$

WLOG assume $m_1 > m_2$ as integers. Subtracting, we obtain:

$$(m_1 - m_2) \cdot T = k_2 - k_1 \pmod q.$$

Since $M \cdot T < q$, there is no wraparound modulo q. Therefore, the above congruence holds over $\mathbb{Z}$ as well. However, since $m_1 \neq m_2$, the left-hand side is at least T, while the right-hand side is strictly less than T because $k_1, k_2 \in [0, T)$. This yields a contradiction.

5 Application: Constraint Programming for Multiset Hashing

5.1 Multiset Hashing and zkVM Memory Consistency

A *multiset hash function* MSH [CDvD+03] is a cryptographic primitive that maps a multiset S over some message space $\mathcal{M}$ to a compact digest, satisfying the following key properties:

- **Compression:** Each multiset is mapped to a fixed-size digest.
- **Incrementality:** Given digests d_1 and d_2 corresponding to multisets S_1 and S_2, one can efficiently compute the digest of their multiset union $S_1 \uplus S_2$.

– **Collision Resistance:** It is computationally infeasible to find two distinct multisets $S_1 \neq S_2$ such that $\mathsf{MSH}(S_1) = \mathsf{MSH}(S_2)$.

A widely used construction paradigm, introduced by Clarke et al. [CDvD$^+$03], builds multiset hash functions from any cryptographic group and a corresponding hash-to-group function.

$$\mathsf{MSH}(S) = \sum_{m \in S} \mathsf{HashToGroup}(m).$$

This construction naturally supports incrementality, as the digest of an updated multiset is simply the sum of the group elements representing the updated entries. Collision resistance can be proven under the random oracle model and the discrete logarithm assumption [BM97]. Shepard et al. [MSTA17] refined this approach specifically for elliptic curves and explored different hash-to-elliptic curve group functions.

Of particular relevance to our work, Brown [Bro08] proposed a heuristic approach that directly maps messages into the group, and conjectured it to be secure:

$$\mathsf{MSH}(S) = \sum_{m \in S} \mathsf{MapToGroup}(m).$$

Looking ahead, in section 5.3, we formalize this approach in the constraint programming context using our map-to-elliptic-curve-group relations, and prove its security in the elliptic curve generic group model. But let us first motivate our work in the context of zkVMs.

The zkVM Memory Consistency Problem. In zero-knowledge virtual machines (zkVMs)such as Jolt, RISC Zero, SP1, or Nexusprograms execute inside a virtual machine while simultaneously generating a cryptographic proof of correct execution. A critical component of this proof is *memory consistency*, which ensures that every read operation retrieves the value from the most recent write to the same address.

A standard approach to enforcing this is the *offline memory checking* paradigm. During execution, the zkVM emits two logs:

– a **write-log** recording every memory write, and
– a **read-log** recording every memory read.

Each log entry encodes a memory access record, typically including a memory address, a data value, a read/write flag, and possibly additional metadata such as a timestamp. To verify memory consistency, at the end of VM execution, we check that the read-log and write-log are equal *as multisets*. Informally, this enforces two sufficient guarantees for memory consistency:

1. Every read matches a corresponding write with the same address and value.
2. Under timestamp ordering, every write is eventually usedi.e., matched by a readso no spurious writes are introduced.

Multiset Hashing for Offline Memory Checking. Rather than bookkeeping the full logswhich is very expensive to maintain throughout VM execution Spice [SAGL18] instead proposed to use a multiset hash function to incrementally compress each log to a short, rolling digest. Thanks to incrementality, each new memory access record can be quickly absorbed into the rolling digest. Eventually, the read-write logs are considered consistent if their final digests match. Collision resistance ensures that no adversary can forge distinct read-write logs that ultimately hash to the same value. Recent work by Succinct takes exactly this approach using a multiset hash function that uses a Poseidon-based hash-to-group function [Roy25] for checking memory consistency in the SP1 zkVM.

5.2 Multiset Hash Relation

In the zkVM setting, every computation must be expressed as a system of constraints. In particular, when dealing with memory checking, this involves constraint programming the use of a multiset hash function to incrementally aggregate over digests of memory accesses. To better handle this situation, we generalize multiset hash functions to a relational setting.

Definition 9 (Multiset Hash Relation). *Let $\mathcal{M}$ be a finite message space,* $\mathsf{Multiset}(\mathcal{M})$ *denote the space of multisets over $\mathcal{M}$, $\mathcal{D}$ a digest space, and $\mathcal{W}$ a witness space. A* multiset hash relation *is a ternary relation:*

$$\mathcal{R}_{\mathcal{MSH}} \subseteq \mathsf{Multiset}(\mathcal{M}) \times \mathcal{D} \times \mathcal{W}$$

satisfying the following properties:

- ***Compression:*** *Every digest $d \in \mathcal{D}$ has a fixed size.*
- ***Incrementality:*** *For any two valid instances $(\mathcal{S}_1, d_1, w_1), (\mathcal{S}_2, d_2, w_2) \in \mathcal{R}_{\mathcal{MSH}}$, let $d = d_1 + d_2$. Then one can efficiently compute some witness w such that*

$$(\mathcal{S}_1 \uplus \mathcal{S}_2, d, w) \in \mathcal{R}_{\mathcal{MSH}},$$

  where $\uplus$ denotes multiset union and $+$ is a constant-time operation in $\mathcal{D}$.
- ***Multiset Collision Resistance:*** *For any efficient adversary $\mathcal{A}$,*

$$\Pr\left[\begin{smallmatrix}(\mathcal{S}_1,\mathcal{S}_2,d,w_1,w_2)\leftarrow\mathcal{A}:\\(\mathcal{S}_1,d,w_1),(\mathcal{S}_2,d,w_2)\in\mathcal{R}_{\mathcal{MSH}}\wedge\mathcal{S}_1\neq\mathcal{S}_2\end{smallmatrix}\right] \leq \mathsf{negl}(\lambda).$$

From the sake of witness generation, we further require that $\mathcal{R}_{\mathcal{MSH}}$ is efficiently constructible*. That is, there exists a deterministic polynomial-time algorithm $\mathsf{Construct}_{\mathcal{R}_{\mathcal{MSH}}} : \mathsf{Multiset}(\mathcal{M}) \to \mathcal{D} \times \mathcal{W}$ satisfying:*

- ***Correctness:*** *For any multiset $\mathcal{S}$, the output $(d, w) = \mathsf{Construct}_{\mathcal{R}_{\mathcal{MSH}}}(\mathcal{S})$ satisfies $(\mathcal{S}, d, w) \in \mathcal{R}_{\mathcal{MSH}}$, except with negligible failure probability.*
- ***Homomorphic Composition:*** *Due to incrementality of the relation, the constructor must be homomorphic with respect to multiset union and digest addition. That is, for any two multisets $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{M}$, if*

$$(d_1, \dots) = \mathsf{Construct}_{\mathcal{R}_{\mathcal{MSH}}}(\mathcal{S}_1), \quad (d_2, \dots) = \mathsf{Construct}_{\mathcal{R}_{\mathcal{MSH}}}(\mathcal{S}_2),$$

  then

$$\mathsf{Construct}_{\mathcal{R}_{\mathcal{MSH}}}(\mathcal{S}_1 \uplus \mathcal{S}_2) = (d_1 + d_2, \dots).$$

5.3 Construction of Multiset Hash Relation

We now construct a multiset hash relation $\mathcal{R}_{\mathcal{MSH}}$ which combines Brown's [Bro08] idea with any map-to-group relation $\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G \times \mathcal{W}$ that is injective, inverse-excluding, and efficiently constructible.

Since the constructor $\mathsf{Construct}_{\mathcal{R}_{\mathcal{MSH}}}$ must be homomorphic, in order to specify it, we just need to define its behavior on the empty multiset and specify an incremental rule for extending the digest and witness with a new element $m \in \mathcal{M}$. These steps are summarized in fig. 3. Notice that the constructor inherits the negligible failure probability from the constructor of $\mathcal{R}_{\mathsf{M2G}}$.

The corresponding multiset hash relation is described in fig. 4.

Incremental Constructor for the Multiset Hash Relation $\mathcal{R}_{\mathcal{MSH}}$

Parameters: Let $\mathsf{Construct}_{\mathcal{R}_{\mathsf{M2G}}} : \mathcal{M} \to G \times \mathcal{W}_G$ be the constructor of the map-to-group relation $\mathcal{R}_{\mathsf{M2G}}$.
Base Case: For the empty multiset $\mathcal{S} = \emptyset$:

- Digest: $d_\emptyset \leftarrow \mathcal{O} \in G$ (infinity point of the group)
- Witness: $w_\emptyset \leftarrow \emptyset$ (the empty **multiset**)

Incremental Step: Given $(d_\mathcal{S}, w_\mathcal{S}) = \mathsf{Construct}_{\mathcal{R}_{\mathcal{MSH}}}(\mathcal{S})$ and new message $m \in \mathcal{M}$:

- Compute $((x_{u_m}, y_{u_m}), w_m) \leftarrow \mathsf{Construct}_{\mathcal{R}_{\mathsf{M2G}}}(m)$
- Update digest: $d_{\mathcal{S}\uplus\{m\}} \leftarrow d_\mathcal{S} + (x_{u_m}, y_{u_m})$
- Update witness: $w_{\mathcal{S}\uplus\{m\}} \leftarrow w_\mathcal{S} \cup \{(m, (x_{u_m}, y_{u_m}), w_m)\}$

Fig. 3: Constructor for the multiset hash relation $\mathcal{R}_{\mathcal{MSH}}$.

Multiset Hash Relation $\mathcal{R}_{\mathcal{MSH}}$

Parameters:

- Let $\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G \times \mathcal{W}_{\mathsf{m2g}}$ be an injective and inverse-excluding map-to-elliptic-curve-group relation.
- Define digest space $\mathcal{D} := G$, and witness space $\mathcal{W}_{\mathsf{mset}} := \mathsf{Multiset}(\mathcal{M} \times G \times \mathcal{W}_{\mathsf{m2g}})$.

Relation $\mathcal{R}_{\mathcal{MSH}} \subseteq \mathsf{Multiset}(\mathcal{M}) \times \mathcal{D} \times \mathcal{W}_{\mathsf{mset}}$:
A triple $(\mathcal{S}, d, w)$ belongs to $\mathcal{R}_{\mathcal{MSH}}$ if and only if:

1. If $\mathcal{S} = \emptyset$, then assert $d = \mathcal{O}$.
2. $\mathcal{S} = \{m_1, \ldots, m_N\}$ be a multiset of messages in $\mathcal{M}$,
3. $w = \{(m_1, (x_{u_{m_1}}, y_{u_{m_1}}), w_{m_1}), \ldots, (m_N, (x_{u_{m_N}}, y_{u_{m_N}}), w_{m_N})\}$, where each $(m_i, (x_{u_{m_i}}, y_{u_{m_i}}), w_{m_i}) \in \mathcal{R}_{\mathsf{M2G}}$,
4. The digest $d = (x, y)$ satisfies:
   $d = (x_{u_{m_1}}, y_{u_{m_1}}) + \cdots + (x_{u_{m_N}}, y_{u_{m_N}})$.

Fig. 4: Description of the multiset hash relation $\mathcal{R}_{\mathcal{MSH}}$.

Theorem 2. *The multiset hash relation $\mathcal{R}_{\mathcal{MSH}}$ defined in fig. 4 satisfies the **compression** and **incrementality** properties of definition 9.*

Proof. By construction, each digest is a single elliptic curve group element. Now let $(\mathcal{S}_1, d_1, w_1), (\mathcal{S}_2, d_2, w_2) \in \mathcal{R}_{\mathcal{MSH}}$, where

$$w_1 = \{(m_i, (x_{u_{m_i}}, y_{u_{m_i}}), w_{m_i})\}_{i=1}^{n}, \quad w_2 = \{(m_j, (x_{u_{m_j}}, y_{u_{m_j}}), w_{m_j})\}_{j=1}^{k}.$$

Define $d := d_1 + d_2$ and $w := w_1 \cup w_2$. It's easy to see that $(\mathcal{S}_1 \uplus \mathcal{S}_2, d, w) \in \mathcal{R}_{\mathcal{MSH}}$.

Theorem 3 (Multiset Collision Resistance in EC-GGM). *Let $\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G \times \mathcal{W}$ be a map-to-elliptic-curve-group relation that satisfies being injective, inverse excluding, and independent of the EC-GGM encoding function $\mathsf{enc} : \mathbb{F}_p \to G$. Let $\mathcal{R}_{\mathcal{MSH}}$ be the multiset hash relation constructed in fig. 4.*

Then for any probabilistic polynomial-time adversary $\mathcal{A}$ making at most Q queries to the EC-GGM addition oracle, the probability that $\mathcal{A}$ finds a multiset collision is at most

$$\frac{2 \cdot \sum_{m \in \mathcal{M}} |S_m^{\pm}| \cdot Q + 2Q^2}{p}.$$

Proof. Suppose the adversary outputs a valid multiset collision:

$$(\mathcal{S}_1, d, w_1), \quad (\mathcal{S}_2, d, w_2) \in \mathcal{R}_{\mathcal{MSH}}, \quad \mathcal{S}_1 \neq \mathcal{S}_2.$$

Let $H_1 = \{(x_{u_{m,w}}, y_{u_{m,w}})\}_{(m,(x_{u_{m,w}}, y_{u_{m,w}}),w) \in w_1}$ be the multiset of group elements associated with witness w_1, and define H_2 similarly for w_2. Define their multiset difference as $H_\Delta = H_1 \uplus \overline{H_2}$, where $\overline{H_2}$ denotes H_2 with negated multiplicitiesi.e., each element g in H_2 appears with count $(-c)$ if it originally appeared c times. Since $\mathcal{S}_1 \neq \mathcal{S}_2$ and $\mathcal{R}_{\mathsf{M2G}}$ is injective, it follows that $H_\Delta \neq \emptyset$.

From the definition of $\mathcal{R}_{\mathcal{MSH}}$, we know:

$$\sum_{(x_{u_m}, y_{u_m}) \in H_\Delta} c_{(x_{u_m}, y_{u_m})} \cdot (x_{u_m}, y_{u_m}) = \mathcal{O},$$

where $c_{(x_{u_m}, y_{u_m})}$ denotes the signed multiplicity in H_Δ. That is, the adversary finds a nontrivial linear combination of group elements summing to zero (infinity point). Since each group element must be in $\mathcal{R}_{\mathsf{M2G}}$ and corresponds to a discrete logarithm $u_m \in \mathbb{F}_p$ where $[u_m] \in S_m^{\pm}$ for some $m \in \mathcal{M}$, this induces a non-trivial linear relation over discrete logs:

$$\sum_{m \in \mathcal{M},\ [u_m] \in S_m^{\pm},\ u_m \in [u_m]} \beta_{u_m} \cdot u_m = 0,$$

for some coefficients $\beta_{u_m} \neq 0$. We now analyze the probability of this event.

Let the adversary make Q queries to the EC-GGM addition oracle. Without loss of generality, assume each query outputs a group element (x_{u_j}, y_{u_j}), corresponding to a discrete log of the form:

$$u_j = \beta_0 + \sum_{m_k \in \mathcal{M},\, [u_{m_k}] \in S^{\pm}_{m_k},\, u_{m_k} \in [u_{m_k}]} \beta_{u_{m_k}} \cdot u_{m_k},$$

with adversarially chosen coefficients $\beta_0, \beta_{u_{m_k}} \in \mathbb{F}_p$.

Event A: Outside Collision. An *outside collision* occurs if an oracle output has discrete log in a class $[u_{m^*}] \in S^{\pm}_{m^*}$ in a nontrivial way (i.e., not through trivial combinations like $3u_{m^*} - 2u_{m^*}$). Since each u_j is a linear combination over independent uniformly random $[u_{m_k}]$, by theorem 1, the probability that $u_j \in [u_{m^*}]$ for any fixed m^* is at most $2/p$. Taking a union bound over all Q queries and all classes $[u_{m^*}] \in S^{\pm}_{m^*}$, we get:

$$\Pr[\mathsf{Event\ A}] \leq \frac{2 \cdot \sum_{m \in \mathcal{M}} |S^{\pm}_m| \cdot Q}{p}.$$

Event B: Inside Collision. This occurs if two oracle outputs (x_{u_i}, y_{u_i}) and (x_{u_j}, y_{u_j}) share the same inversion class $[u_i] = [u_j]$. By a similar argument, the probability that any two such queries collide is at most $2/p$. There are $\binom{Q}{2} \leq Q^2$ such pairs, so:

$$\Pr[\mathsf{Event\ B}] \leq \frac{2Q^2}{p}.$$

Suppose neither event occurs. Then all discrete logs variables used in H_Δ must have no non-trivial correlations (except for the linear relations already known to the adversary) and thus are independently and uniformly random over $\mathbb{F}^{\pm}_p$ by theorem 1. Since $\mathcal{R}_{\mathsf{M2G}}$ is inverse excluding, there is at most one representative per class, and thus:

$$\Pr\left[\sum \beta_{u_m} \cdot u_m = 0\right] \leq \Pr\left[\sum \beta_{[u_m]} \cdot [u_m] = 0\right] \leq \frac{2}{p}.$$

We then conclude the proof by taking union bound over all above.

5.4 Concrete Parameters for zkVM Application

In our target applicationenforcing memory consistency in zkVMsthe multiset hash relation $\mathcal{R}_{\mathcal{MSH}}$ operates over memory access records. Each message $m \in \mathcal{M}$ corresponds to a memory access log entry. For example, in a 32-bit RISC-V machine, a typical record includes:

- a 32-bit memory address,
- 32-bit data,
- a 1-bit read/write flag,
- optional metadata, such as a 32-bit timestamp.

These fields total approximately 97 bits, so the message space satisfies $|\mathcal{M}| \leq 2^{100}$.

Concretely, we instantiate the multiset hash relation where the underlying map-to-elliptic-curve-group relation is defined in fig. 2 with its constructor defined in fig. 1. We fix the tweak bound $T = 256$, which ensures that the constructor fails with probability at most $N \cdot 2^{-256}$, where N denotes the total number of memory accesses (reads and writes) in one zkVM execution.

Since each message $m \in \mathcal{M}$ is associated with at most one of T possible elliptic curve points, the number of discrete log inversion classes satisfies $|S^{\pm}_m| \leq T = 2^8$. Applying theorem 3, we retain well over 120 bits of security against multiset collisions.

6 Application: Constraint Programming for BLS Signing

6.1 Proof of Stake and zkPoS

Proof of Stake (PoS) protocols such as Ethereum PoS rely on cryptographic signatures for validating blocks and aggregating attestations. In such contexts, [BLS04] (BLS) signatures are widely used due to their algebraic nature, which allows multiple signatures over the same block to be efficiently aggregated into a single one.

Zero-Knowledge Proof of Stake (zkPoS) [Hyp23] refers to a cryptographic framework where the consensus mechanism of a PoS blockchain is proven in zero knowledge. Rather than trusting honest majority assumptions or relying on full nodes, a verifier (e.g., a light client or smart contract) can check succinct zero-knowledge proofs that attest to the validity of many PoS consensus steps. In particular, this includes verifying that many different blocks were signed by a sufficient quorum of validators, which incurs zero-knowledge verification of multiple BLS signatures.

6.2 Relational Signature Scheme

In the constraint programming paradigm, we adopt a relational view of signature schemes. Instead of representing verification as a deterministic algorithm, we observe that signature validity can be naturally expressed as a relation between the message, signature, and verification key. This motivates the notion of a *relational signature scheme*, where verification is defined by a relation rather than a deterministic function.

Definition 10 (Relational Signature Scheme). *A* relational signature scheme *consists of:*

- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*: A probabilistic algorithm that outputs a secret and verification key.*
- $(\sigma_m, w_m) \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$*: An algorithm that produces a signature σ_m as well as some auxiliary witness info on message m.*
- $\mathcal{R}_{\mathsf{Sig}} \subseteq \mathcal{VK} \times \mathcal{M} \times \Sigma \times \mathcal{W}$*: A relation which decides whether a tuple $(\mathsf{vk}, m, \sigma_m)$ is valid, which may leverage some auxiliary witness w_m.*

A relational signature scheme must satisfy the following properties:

- ***Correctness:***

$$\Pr\left[(\mathsf{vk}, m, \sigma_m, w_m) \in \mathcal{R}_{\mathsf{Sig}} \;\middle|\; \begin{array}{l}(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda),\\ m \leftarrow \mathcal{M},\\ (\sigma_m, w_m) \leftarrow \mathsf{Sign}(\mathsf{sk}, m)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

- ***Unforgeability (EUF-CMA):*** *For all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following game is negligible in λ:*
  1. *The challenger runs $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.*
  2. *The adversary $\mathcal{A}$ is given vk and access to a signing oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$.*
  3. *For each query m, the oracle returns $(\sigma_m, w_m) \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$.*
  4. *Eventually, $\mathcal{A}$ outputs a forgery $(m^*, \sigma_{m^*}, w_{m^*})$.*
  5. *$\mathcal{A}$ wins if:*
     - *m^* was never queried to the signing oracle, and*
     - *$(\mathsf{vk}, m^*, \sigma_{m^*}, w_{m^*}) \in \mathcal{R}_{\mathsf{Sig}}$.*

6.3 Relational BLS Signature from Map-to-Elliptic-Curve-Group Relation

We describe a constraint-friendly relational signature scheme based on BLS signature scheme. The high level idea is to replace its hash-to-group function with our map-to-elliptic-curve-group relation. The full description of the scheme is shown as in fig. 5.

Theorem 4 (Correctness). *Let $\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G_1 \times \mathcal{W}$ be the map-to-elliptic-curve-group relation, and let $\mathsf{Construct} : \mathcal{M} \to G_1 \times \mathcal{W} \cup \{\perp\}$ be the associated constructor. The relational BLS signature scheme in fig. 5 satisfies correctness.*

Proof. By efficient constructibility of $\mathcal{R}_{\mathsf{M2G}}$, the constructor $\mathsf{Construct}(m)$ succeeds with probability at least $1 - \mathsf{negl}(\lambda)$, returning (h_m, w_{h_m}) such that $(m, h_m, w_{h_m}) \in \mathcal{R}_{\mathsf{M2G}}$. In such case, due to correctness of BLS signature, the resulting signature must be in the relation $\mathcal{R}_{\mathsf{Sig}}$. Thus, the overall success probability is at least $1 - \mathsf{negl}(\lambda)$.

Since this application uses elliptic curve groups equipped with a bilinear map, to prove unforgeability, we will work with an extended generic group model called the *semi-generic group model (SGGM)* introduced by Jager and Ruprai [JR10], which models the two source groups $G_1,\ G_2$ as generic groups, with an explicit bilinear map oracle which maps $G_1,\ G_2$ to a standard model of G_T. The high-level structure of our unforgeability proof in SGGM parallels that of theorem 3, relying on bounding the probabilities of *inside* and *outside* collisions. Due to space limits, the full proof is deferred to the extended version of this paper.

Relational BLS Signature Scheme

Parameters:

- A bilinear group (G_1, G_2, G_T) of prime order p, with pairing $e : G_1 \times G_2 \to G_T$.
- A map-to-elliptic-curve-group relation $\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G_1 \times \mathcal{W}$ and its associated constructor $\mathsf{Construct}_{\mathsf{M2G}} : M \to G_1 \times \mathcal{W} \cup \{\perp\}$.

$\mathsf{KeyGen}(1^\lambda)$:

- Sample $\mathsf{sk} \leftarrow \mathbb{Z}_p$, then set $\mathsf{vk} := \mathsf{sk} \cdot g_2 \in G_2$, where $g_2 = (x_1', y_1')$ is a fixed generator of G_2

$\mathsf{Sign}(\mathsf{sk}, m)$:

- Run $\mathsf{Construct}_{\mathsf{M2G}}(m) = (h_m, w_{h_m})$ where h_m denotes the element $(x_{u_m}, y_{u_m}) \in G_1$.
- If $(h_m, w_{h_m}) \neq \perp$ then output $w_m = (h_m, w_{h_m})$ and $\sigma_m := \mathsf{sk} \cdot h_m \in G_1$.

$\mathcal{R}_{\mathsf{Sig}}$:

- A tuple $(\mathsf{vk}, m, \sigma_m, w_m)$, where $w_m = (h_m, w_{h_m})$, belongs to $\mathcal{R}_{\mathsf{Sig}}$ if and only if:

$$(m, h_m, w_{h_m}) \in \mathcal{R}_{\mathsf{M2G}} \quad \text{and} \quad e(\sigma_m, g_2) = e(h_m, \mathsf{vk}).$$

Fig. 5: Relational BLS signature scheme using a map-to-elliptic-curve-group relation

Theorem 5 (Unforgeability in SGGM). *Let $\mathcal{R}_{\mathsf{M2G}} \subseteq \mathcal{M} \times G_1 \times \mathcal{W}$ be a map-to-elliptic-curve-group relation that is injective and independent of the SGGM encoding function. Then for any probabilistic polynomial-time adversary $\mathcal{A}$ making Q_1 signing queries and Q_2 addition oracle and bilinear map oracle queries in SGGM, the probability that $\mathcal{A}$ wins the unforgeability game is at most*

$$4 \cdot \frac{\sum_{m \in \mathcal{M}} |S_m^{\pm}| \cdot (Q_1 + Q_2) + (Q_1 + Q_2)^2}{p}.$$

6.4 Concrete Parameters for zkPoS Application

We instantiate the relational BLS signature scheme using our map-to-elliptic-curve-group relation and its constructor described in figs. 1 and 2. By setting the tweak bound $T = 128$, we ensure that for any message m, the constructor succeeds with all but probability less than 2^{-128}. Thus, the relation-based signature scheme can produce valid signatures for all messages except with negligible failure.

Applying theorem 5, we obtain unforgeability guarantees in the EC-GGM model. In particular, if each message can be specified with less than 120-bits, the resulting scheme achieves at least 120-bit security in the unforgeability game.

However, in zkPoS applications, the messages to be signed (e.g., full blocks) may be substantially larger. For example, an Ethereum block may be several kilobytes in size, far exceeding the allowable domain size for our relation.

To support signing arbitrary larger messages $M \in \{0,1\}^*$, we proceed as follows. First, decompose M into ℓ chunks $(m_1, \ldots, m_\ell)$, where each chunk is less than 120 bits and thus can be embedded into some prime field $\mathbb{F}_{p_{\mathsf{msg}}}$ of size around 2^{120}. In the constraint programming setting, this is enforced by chunk-wise decomposition and range constraints ensuring each $m_i < p_{\mathsf{msg}}$.

Next, we define a compressed message representative using a *random curve combination*. Specifically, we introduce a constraint that expresses:

$$\tilde{m} := m_1 + r \cdot m_2 + \cdots + r^{\ell-1} \cdot m_\ell \in \mathbb{F}_{p_{\mathsf{msg}}},$$

where $r \in \mathbb{F}_{p_{\mathsf{msg}}}$ is a random verifier challenge sampled after the prover commits to $(m_1, \ldots, m_\ell)$. In the setting of zkPoS, this randomness can be introduced using FiatShamir over any commit-and-prove scheme. It's easy to

check that if $M^0 = (m_1^0, \ldots, m_\ell^0)$ and $M^1 = (m_1^1, \ldots, m_\ell^1)$ are two different messages, then their respective representatives $(\tilde{m}^0, \tilde{m}^1)$ collide with probability $\leq \ell / p_{\mathsf{msg}}$ over the randomness of r. Concretely, any two 4KB-size full blocks collide with probability $\leq 2^{-110}$.

Finally, the constraint system will verify that the relational signature constraints are satisfied with respect to the compressed representative $\tilde{m}$.

7 Evaluation

In this section, we aim to concretely evaluate the performance of our proposed map-to-elliptic-curve-group relation, as a constraint-friendly replacement for traditional hash-to-elliptic-curve-group functions.

In particular, we focus on the task of memory consistency checking in zkVMs, as described in section 5, where multiset hashing must be repeatedly applied to large read and write logs. Our goal is to compare the performance and constraint cost of proving, within a constraint programming framework, the correctness of multiset hash digests constructed using either:

1. Our *map-to-elliptic-curve-group relation* as described in fig. 2. When instantiated within constraint programming, it yields the multiset hash relation shown in fig. 4.
2. A *hash-to-elliptic-curve-group* function using the increment-and-check methodthe most efficient non-deterministic approach in practice, as discussed in section 1.3. The corresponding multiset hash relation can be defined analogously, using a canonical hash-to-elliptic-curve-group relation $\mathcal{R}_{\mathsf{H2G}}$ in place of $\mathcal{R}_{\mathsf{M2G}}$ in item 3 of fig. 4.

To better isolate and compare the cost of the hash/map-to-elliptic-curve-group step in this application, we benchmark only the portion of the multiset hash relation responsible for mapping or hashing each message in the multiset to a group element $(x, y) \in G$. In particular, we do *not* measure the cost of the final group addition over all mapped points, as this step is identical across both approaches and does not impact the relative comparison.

Parameters. We instantiate our map-to-elliptic-curve-group relation using the configuration described in section 5.4, where the message space $\mathcal{M}$ corresponds to 100-bit memory access records. We fix the tweak bound to $T = 256$, which ensures a negligible failure probability across all invocations. Combined with injectivity and inverse-exclusion properties, this yields over 120 bits of security against multiset collisions, as shown in theorem 3.

For our hash-to-elliptic-curve-group baseline, we benchmark three widely used cryptographic hash functions to assess their constraint costs:

- **SHA-256:** The NIST-standardized hash function with well-established security, but high constraint complexity due to its bit-oriented structure.
- **MiMC:** A non-standard but constraint-friendly hash function designed for zero-knowledge proof systems, configured with exponent 5 and 110 rounds.
- **Poseidon:** Another non-standard constraint-friendly hash, instantiated with state width $t = 3$, rate $r = 2$, 8 full rounds, and 57 partial rounds.

These configurations are selected to achieve approximately 128-bit security and serve as fair baselines for our experiment.

Choice of Elliptic Curve and Field for Constraint Programming. Throughout our evaluation, we fix the elliptic curve group to be BN254, a pairing-friendly curve widely used in zkSNARK applications. As discussed in example 1, constraint systems are defined over a fixed finite field $\mathbb{F}$, which is often dictated by the backend proving system. This introduces a key implementation decision when expressing the underlying elliptic-curve-group operations in constraints: whether to perform arithmetic directly over the elliptic curves scalar field (i.e., a *native field* instantiation), or to emulate it within a different field (i.e., a *non-native field* setting). For comprehensiveness, we evaluate both approaches in our benchmarks.

In the native setting, the elliptic curve group and the constraint field share the same base field. For our benchmarks, we adopt the default configuration of `Noir` and its backend `Barretenberg`, where circuits are compiled over the Grumpkin fielda 254-bit prime field that also serves as the scalar field of BN254. This alignment ensures that elliptic curve operations and constraint logic remain within a unified algebraic environment, avoiding costly field emulation and enabling efficient circuit synthesis.

In the non-native setting, we emulate the BN254 G_1 group operation within `Noir` by representing each elliptic curve point as an array of eight 32-bit field elements. This effectively decomposes 256-bit coordinates into limbs compatible with smaller prime fields, such as the FFT-friendly *Goldilocks field* $\mathbb{F}_{2^{64}-2^{32}+1}$ used in proof systems like Plonky3.

For hash functions, we note that MiMC and Poseidon are field-agnostic and can be implemented directly over any prime field. In contrast, SHA-256 is inherently tied to $\mathbb{F}_{2^{32}}$, so we use the standard non-native implementation and convert the resulting digest into a field element by interpreting the output as a little-endian byte array.

7.1 Experimental Setup

Proof system pipeline We conduct our experiments using the Noir zero-knowledge proof programming framework, whose `nargo` CLI parses, type-checks, and compiles high-level `.nr` programs into an Abstract Circuit Intermediate Representation (ACIR), and can execute the circuit to emit both the ACIR JSON and a compressed witness file. ACIR is backend-agnostic: although Noir defaults to Aztecs Barretenberg, the same IR can be further compiled for other PLONK- or R1CS-based provers (e.g., Arkworks Marlin) without modifying source code significantly. Proving backendsinstalled separately (e.g., via the tool `bbup`)handle CRS generation, proof creation, verification, and verifier export. We used Aztec's Barretenberg backend at version 0.76.0 and the `nargo` CLI at version `1.0.0-beta.3`, which implements the PLONK prover by default.

Tooling limitations Barretenberg supports up to 2^{23} constraints, so we omit any measurements beyond that. Poseidon and MiMC exceed this limit at 2^{16} iterations, which is why Figures 6 and 7 are truncated at 2^{15}. SHA-256 stops at 2^{11} iterations, since 4096 runs already require 16.7 million constraints.

Environment All experiments ran on an AWS c5.18xlarge VM (Intel Xeon Platinum, 36 physical cores/72 vCPUs, 144 GiB RAM, 32 GiB EBS NVMe SSD, Amazon Linux 2023.7.20250428). For each parameter set, we perform an untimed warm-up and then ten timed iterations. We call Pythons `time.time_ns()` immediately before and after each stage (witness execution, proof generation, verification), compute elapsed seconds, and report the mean steady-state latency.

7.2 Performance

In this section, we consider the performance of the native map-/hash-to-elliptic-curve-group implementations, under the default pipeline using Noir with backend Barretenberg. We also consider the constraint costs of both native and non-native instantiations.

Execution time (Noir witness generation) Figure 6 plots the time taken by `nargo execute` as we vary the number of iterations of our map-to-curve primitive in the circuit. As the workload scales, we see that all hash-based constructions require higher execution time than map-to-curve. However, while SHA-256 is the worst performing hash function in terms of constraint costs, due to its performance on CPU, the CPU-native witness generation code runs much more efficiently for SHA-256, than for the arithmetic hash functions we tested. We were able to obtain a larger range of results for map-to-curve, due to its constraint-friendliness, and the execution time of map-to-curve for over 250k iterations was about 19s, orders of magnitude better than arithmetic hash-to-elliptic-curve-group for $1/8$th the number of iterations.

Proving time Figure 7 illustrates the proving time of each of our map-/hash-to-elliptic-curve-group instantiations at the number of constraints grow. Unlike in the case of execution time, SHA-256 performs significantly worse on proving time due to its high arithmetic circuit constraint cost. The proving time of 2^{18}, i.e. more than 250k iterations of map-to-curve averages to about 47s, again, performing more than an order of magnitude better than any other hash function at its highest tested number of iterations.

Verification time We omit a detailed graph of verification times, since the growth in verification times is relatively slow. For instance, the verification time for the SHA-256-based construction is ≈ 0.098s when proving 2 executions of hash-to-elliptic-curve-group and 0.101 when proving 2048. At the proof of 2^{15} iterations, Poseidon, which is the worst construction we measured at this size takes 0.121s to verify.

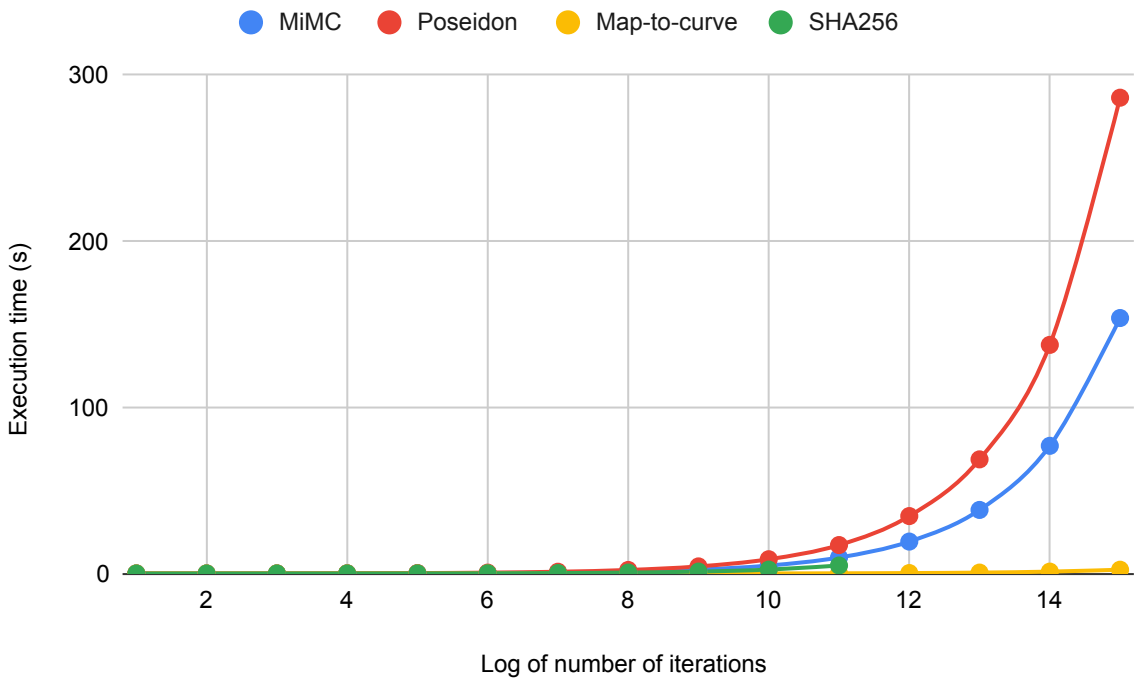

Fig. 6: Noirlang's execution time to compile and build the witnesses for different instantiations of hash or map-to-elliptic-curve-group as the number of iterations of this operation increases.

Constraint costs In table 2, we present the PLONK constraints for native and non-native implementations, as obtained by compiling the Noir program for checking the provided witnesses and using Barretenberg's `bb gates` command to obtain the number of PLONK constraints. For each setting (map-to-elliptic-curve-group and hash-to-elliptic-curve-group for each of our hash functions), we provide the constraints for a single iteration of hash/map-to-elliptic-curve-group. We also provide the number of constraints for 2^{10} iterations in the native case and 2^8 iterations in the non-native case.

Instantiation	**Native impl.: # constraints for 1 iteration**	**Native impl.: # constraints for 2^{10} iterations**	**Non-native impl.: # constraints for 1 iteration**	**Non-native impl.: # constraints for 2^8 iterations**
Map-to-group relation	21+9*	5,136+9,216**= 14,352	57,943+9*	14,130,372+2,304**
MiMC hash-to-group	351	343,056	58,273	14,214,852
Poseidon hash-to-group	948	954,384	58,870	14,367,684
SHA-256 hash-to-group	7,095	4,196,023	62,276	15,177,591

Table 2: PLONK constraint costs of the map-to-elliptic-curve-group relation and hash-to-elliptic-curve-group using different hash functions, for both native and non-native implementations. Note that due to the large constraint size non-native iterations of map and hash-to-elliptic-curve-group and the expected linear growth in circuit size, we capped our tests for non-native constraint costs at 2^8 iterations. *This is the number of PLONK (addition and non-constant-multiplication) constraints for an 8 bit range check. Noirlang requires certain typecasting, which leads to a blowup in the number of constraints for range checking, hence we only report the range checking numbers analytically. **The 9,216 constraints are added by 2^{10} range checks and $2,304$ by 2^8 range checks.

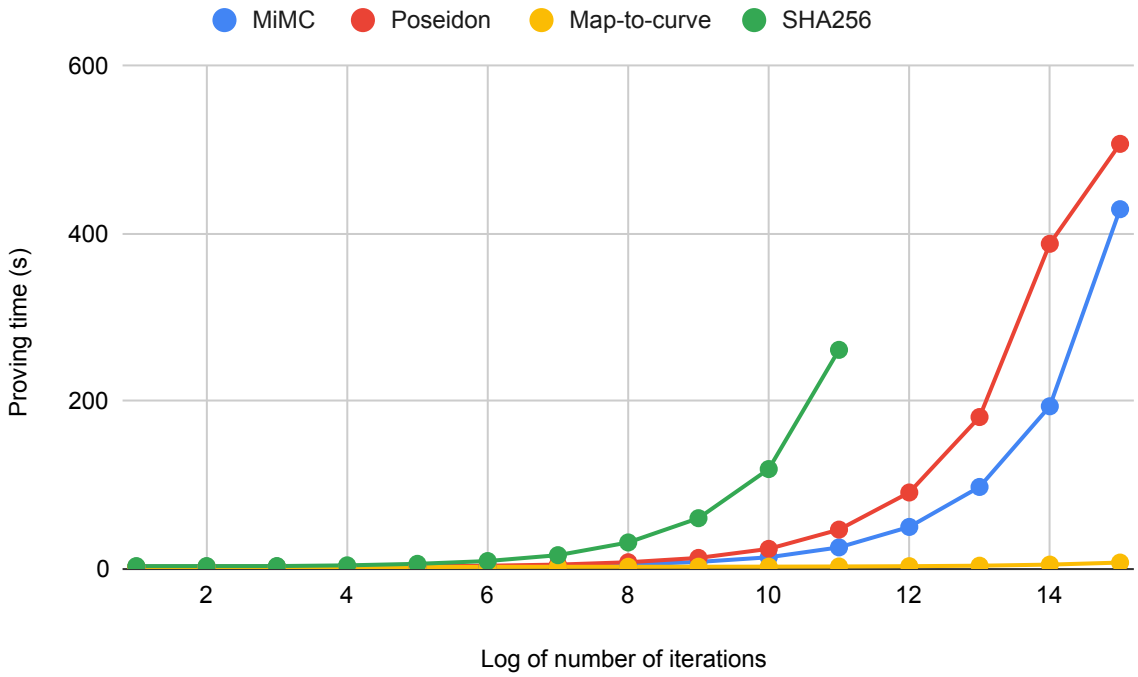

Fig. 7: Barretenberg's proving time for different instantiations of hash or map-to-elliptic-curve-group as the number of iterations of this operation increases. Note that this operation uses as input the circuit description and witnesses provided by the Noirlang execution from fig. 6.

Acknowledgments. The authors thank the anonymous reviewers for their valuable feedback.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

ACG+25. Michel Abdalla, Arka Rai Choudhuri, Jens Groth, Yoichi Hirai, Ben Hoberman, Samuel Judson, Daniel Marin, Victor S. Miller, Duc Tri Nguyen, Evan Schott, and Kristian Sosnin. Nexus zkvm 3.0 specification. Technical report, Nexus, March 2025. Revised March 11, 2025.

AGR+16. Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 191–219. Springer, 2016.

AST24. Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. In *Advances in Cryptology EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 2630, 2024, Proceedings, Part VI*, page 333, Berlin, Heidelberg, 2024. Springer-Verlag.

BCI+10. Eric Brier, Jean-Sebastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010, Proceedings*, volume 6223, page 237. Springer Science & Business Media, 2010.

BEG+91. Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, SFCS '91, page 9099, USA, 1991. IEEE Computer Society.

BHKL13. Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 967–980, 2013.

BLS04. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of cryptology*, 17(4):297–319, 2004.

BM97. Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 163–192. Springer, 1997.

Bro08. Daniel R. L. Brown. The encrypted elliptic curve hash. *IACR Cryptol. ePrint Arch.*, page 12, 2008.

CDvD+03. Dwaine Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G Edward Suh. Incremental multiset hash functions and their application to memory integrity checking. In *Advances in Cryptology-ASIACRYPT 2003: International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30-December 4, 2003, Proceedings*, volume 2894, page 188. Springer Science & Business Media, 2003.

CSRHT22. Jorge Chavez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Swiftec: Shallue-van de woestijne indifferentiable function to elliptic curves: Faster indifferentiable hashing to elliptic curves. In *Advances in Cryptology ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 59, 2022, Proceedings, Part I*, page 6392, Berlin, Heidelberg, 2022. Springer-Verlag.

FFS+13. Reza R. Farashahi, Pierre-Alain Fouque, Igor E. Shparlinski, Mehdi Tibouchi, and Jos F. Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. *Mathematics of Computation*, 82(281):491–512, 2013.

FHSS+23. A. Faz-Hernandez, S. Scott, N. Sullivan, R. S. Wahby, and C. A. Wood. Rfc 9380: Hashing to elliptic curves, 2023.

FJT13. Pierre-Alain Fouque, Antoine Joux, and Mehdi Tibouchi. Injective encodings to elliptic curves. In *Information Security and Privacy ACISP 2013*, volume 7959 of *Lecture Notes in Computer Science*, pages 203–218. Springer, Heidelberg, 2013.

FSV09. Reza R. Farashahi, Igor E. Shparlinski, and José Felipe Voloch. On hashing into elliptic curves. *Journal of Mathematical Cryptology*, 3(4):353–360, December 2009.

FT10. Pierre-Alain Fouque and Mehdi Tibouchi. Deterministic encoding and hashing to odd hyperelliptic curves. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *Pairing-Based Cryptography – Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 265–277. Springer, 2010.

GKR+21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535, 2021.

GS22. Jens Groth and Victor Shoup. On the security of ecdsa with additive key derivation and presignatures. In *Advances in Cryptology EUROCRYPT 2022: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 June 3, 2022, Proceedings, Part I*, page 365396, Berlin, Heidelberg, 2022. Springer-Verlag.

Hyp23. Hyper Oracle Team. zkpos with halo2-pairing for verifying aggregate bls signatures. `https://ethresear.ch/t/zkpos-with-halo2-pairing-for-verifying-aggregate-bls-signatures/14671`, January 2023. Accessed: 2025-05-15.

Ica09. Thomas Icart. How to hash into elliptic curves. In *Advances in Cryptology - CRYPTO 2009*, pages 303–316, 2009.

JR10. Tibor Jager and Andy Rupp. The semi-generic group model and applications to pairing-based cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 539–556. Springer, 2010.

Kos23. Dmitrii Koshelev. Hashing to elliptic curves over highly 2-adic fields $\mathbb{F}_q$ with $o(\log(q))$ operations in $\mathbb{F}_q$. Cryptology ePrint Archive, Paper 2023/121, 2023.

Kos24. Dmitrii Koshelev. Hashing to elliptic curves through cipollalehmermüllers square root algorithm. *J. Cryptol.*, 37(2), February 2024.

Kos26. Dimitri Koshelev. Simultaneously simple universal andindifferentiable hashing toelliptic curves. In Abderrahmane Nitaj, Svetla Petkova-Nikova, and Vincent Rijmen, editors, *Progress in Cryptology - AFRICACRYPT 2025*, pages 395–413, Cham, 2026. Springer Nature Switzerland.

Lab25. Succinct Labs. Sp1 introduction. `https://docs.succinct.xyz/docs/sp1/introduction`, 2025. Accessed: 2025-09-09.

MSTA17. Jeremy Maitin-Shepard, Mehdi Tibouchi, and Diego F Aranha. Elliptic curve multiset hash. *The computer journal*, 60(4):476–490, 2017.

RIS25. RISC Zero. Risc zero zkvm api documentation. `https://dev.risczero.com/api/zkvm`, 2025.

Roy25. Uma Roy. Sp1 turbo: the worlds fastest zkvm just got faster. `https://blog.succinct.xyz/sp1-turbo/`, 2025. Accessed: 2025-09-09.

SAGL18. Srinath T. V. Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 339–356. USENIX Association, 2018.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 256–266, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

Ska05. Marlusz Skaba. Points on elliptic curves over finite fields. *Acta Arithmetica*, 117(3):293–301, 2005.

SvdW06. Andrew Shallue and Christiaan van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *International Algorithmic Number Theory Symposium*, pages 510–524, 2006.

TK17. Mehdi Tibouchi and Taechan Kim. Improved elliptic curve hashing and point representation. *Designs, Codes and Cryptography*, 82(1-2):161–177, 2017.

Ula07. Maciej Ulas. Rational points on certain hyperelliptic curves over finite fields. *Bulletin of the Polish Academy of Sciences. Mathematics*, 55(2):97–104, 2007.

WB19. Riad S Wahby and Dan Boneh. Fast and simple constant-time hashing to the bls12-381 elliptic curve. *Cryptology ePrint Archive*, 2019.