

Garbling Groth16 with Native Group Operations

Nakul Khambhati^{*1}, Aaron Feickert^{†2}, Christian Lewe², and Mukesh Tiwari²

¹University of California, Los Angeles

²Alpen Labs

Abstract

The Groth16 verification equation has a compact algebraic description, yet garbling its Boolean implementation can require tens of gigabytes. We show how to garble this computation using native elliptic-curve group operations. Motivated by proof verification in trust-minimized Bitcoin bridges, we construct a projective partial garbling scheme for conditional disclosure on invalid Groth16 proofs. For a fixed verification key and public statement, evaluation reveals a garbler-held secret when the supplied proof is invalid and hides the secret when it is valid. The proof itself remains public.

The main technical challenge is handling a pairing whose two arguments are supplied by the evaluator. We introduce a rekeying gadget that resolves this using a single private scalar multiplication. Building on Argo MAC (Eagen and Lai, 2026), BABE (Garg et al., 2026), and Duty-Free Bits (Khambhati et al., 2026), we prove a composition theorem for partial garbling across input representations and apply it to transform group encodings into bitwise projective encodings. This requires extending the projectivization technique of Duty-Free Bits to quadratic extension fields. The resulting garbled program has size $O(\lambda^2)$ bits, where λ is the computational security parameter. We prove privacy under the decisional Diffie–Hellman assumption, in the random oracle model.

Our BN254 implementation produces a 2.4 MiB garbled program, including projectivization, compared with the 48 GB reported for Boolean garbling of Groth16. Thus, we improve over prior state of the art in Groth16 garbling by over four orders of magnitude.

1 Introduction

Garbling schemes are versatile cryptographic primitives that enable, among other things, computations to be securely outsourced while preserving the privacy of the underlying data. The cost of garbling a computation depends heavily on how that computation is represented. Yao’s original scheme [26] requires one to write the computation as a Boolean circuit. A long line of work has since reduced the size of the resulting garbled program, which ends up being the main bottleneck when using these schemes in practice. Arithmetic garbling exploits a different structure: by operating directly on ring elements, it avoids the cost of expressing arithmetic as a Boolean circuit [3, 6, 15, 5, 20]. For some computations, however, there is useful structure beyond the underlying field arithmetic. A verifier for a pairing-based SNARK is one such example. Its verification equation consists of a few operations on group elements, but implementing those operations as a

^{*}Corresponding author: nakul@cs.ucla.edu

[†]Corresponding author: aaron@alpenlabs.io

Boolean circuit can require billions of gates [8, 2]. In this work, we study how to garble such a computation natively using group operations themselves.

Our key motivating application is proof verification in trust-minimized Bitcoin bridges. Bitcoin’s scripting language does not support verification of SNARKs. Recent protocols address this limitation by moving verification off chain and using the outcome to enforce a spending condition on chain [22, 10, 25]. In the garbling-based approach, an operator prepares a garbled verifier before the proof is available. Later, the operator publishes a proof together with Lamport signatures on its bits. These signatures authenticate the input encoding selected by each proof bit. If the proof is invalid, evaluation reveals a secret that a challenger can use to contest the operator’s claim. If the proof is valid, the secret remains hidden. This reduces the on-chain work to checking signatures and spending conditions, while the challenger evaluates the verifier locally. For this application, we typically require maliciously secure garbling. We present our protocol as a garbling scheme, which is most naturally instantiated as a semi-honest protocol, but note that cut-and-choose can be used to upgrade to malicious security as done in prior work [13, 18].

The resulting off-chain cost can be substantial. Groth16 [14], a common choice in this setting, has proofs consisting of three group elements and a verification equation involving pairings. Compiling this verifier into a Boolean circuit, even with efficient privacy-free garbling, produces garbled material on the order of tens of gigabytes [13, 18]. Each field multiplication inside a pairing becomes a large Boolean subcircuit, and the algebraic relations among these operations are lost. The resulting cost in setup communication and storage has motivated a recent line of work on more compact garbled verifiers.

Algebraic approaches. Argo MAC [11] introduces encodings that preserve some of the verifier’s group structure. A point V is encoded as $\phi_c(V) + K$, where ϕ_c is a secret endomorphism chosen from a small set and K is a secret masking point. Following [11], the secret offset K may be viewed as a MAC key. In the affine-encoding view used in this paper, the same element acts as an additive mask. We may interchangeably use these terms. A vector of such encodings forms an Argo MAC. These encodings support addition, scalar multiplication by private bits, and pairing against a fixed group element without requiring additional garbled material. Eagen and Lai also show how to produce the encodings from a point’s coordinates using the information-theoretic partial garbling techniques of Ishai and Wee [16]. BABE [13] builds on these techniques to garble multiplication of an evaluator-supplied point by an arbitrary private scalar, and combines this with witness encryption for linear pairing relations [12] to design a custom Groth16 verification protocol on Bitcoin.

These information-theoretic partial garbling schemes themselves have encoding functions that are affine functions over large finite fields. Using these algebraic encodings in a Bitcoin protocol requires a further step. Lamport signatures, and other compatible signature schemes suggested in recent works [24, 23], allow the selected encoding string for each input bit to be authenticated. The input encoding must therefore be made *projective*: each bit independently selects one of two fixed encoding strings. Duty-Free Bits [17] gives an efficient transformation from affine encodings over prime fields to projective encodings. Its cost is proportional to the input and output encodings, substantially reducing the overhead of applying the algebraic techniques in this setting.

Our results. We construct a projective partial garbling scheme for conditional disclosure on invalid Groth16 proofs. We fix the verification key and public statement at garbling time. We note that this setting suffices for certain bridge designs, but one could extend our techniques to accept a variable public statement too. In the setting we consider, a garbler holds some high-entropy secret $s \in \{0, 1\}^\lambda$, and the evaluator holds a Groth16 proof $\pi = (A, B, C)$ in the clear. Given the garbled

program and an encoding of π , the evaluator recovers s if the proof is invalid and learns nothing about s if it is valid. We use the partial-garbling formulation of Ishai and Wee [16], in which only a part of the input (i.e., the garbler’s input) is hidden from the evaluator. For a disclosure function, privacy captures the required protection of the secret on valid proofs. The final encoding consists of one of two λ -bit encoding strings per input bit, as required by the Bitcoin application and by standard oblivious-transfer-based implementations of two-party computation.

Our construction evaluates the verification equation through native group operations. The main difficulty is in handling pairings $e(A, B)$, whose two arguments are both supplied by the evaluator. We construct a *rekeying gadget* for garbled pairings and combine it with existing techniques for encoding points and projectivizing affine encodings. Our contributions are as follows.

1. We construct Π_G , a partial garbling scheme for Groth16 secret disclosure with an input encoding that is affine over groups. Its rekeying gadget handles the pairing of two evaluator inputs using a single private scalar multiplication. The resulting garbled program is tiny, consisting of $O(\lambda)$ group elements and commitments, where λ is a computational security parameter. We prove privacy under DDH in the random oracle model (Section 5).
2. We state and prove a composition theorem for partial garbling across changes in the evaluator’s input representation (Section 6). We apply it to pass from group elements to field coordinates and then to bits, incorporating validity checks when a representation does not define a valid input. This gives a common framework for the encoding transformations used in Argo MAC [11], BABE [13], and Duty-Free Bits [17].
3. We extend the projectivization technique of Duty-Free Bits to affine encodings over $\mathbb{F}_{p^2}$, as needed for the coordinates of the proof element B . Together with the composition theorem, this yields a projective scheme Π_{bin} for Groth16 garbling (Section 7).

Concrete cost. Our BN254 implementation produces a garbled program of size 2.4 MiB, which includes the cost of projectivization. Section 8 gives the implementation details and cost breakdown. This is approximately $20,000\times$ smaller than the 48 GB garbled program reported by Mosaic for privacy-free garbling of a Boolean Groth16 verifier [18]. The complex-multiplication optimization described in Appendix A is estimated to reduce the program size further to approximately 1.9 MiB.

Relation to prior work. Eagen and Lai explicitly motivate Argo MAC [11] by the goal of constructing garbled verifiers for pairing-based SNARKs. Their paper gives the encoding primitive and estimates the cost of such a verifier, while leaving the verifier construction, and in particular, an efficient instantiation of the required pairing gadget, to subsequent work. Our protocol was developed independently of this subsequent work, which, to the best of our knowledge, is not public. In this paper, we provide an explicit construction and security proof, including the rekeying gadget for pairings, together with a composition argument that projectivizes the scheme. The group encodings and their efficient realization from coordinates are due to Argo MAC [11], with the projectivization improvements of Duty-Free Bits [17]. Further, we note that BABE [13] also uses the idea of garbled private scalar multiplication to handle pairings within a witness encryption framework. We remark that these techniques do not imply our result, as BABE realizes a different functionality through witness encryption and does not provide a method to handle pairings within a partial-garbling framework. Further, BABE’s bridge design restricts challenges to a designated verifier set, whereas our construction is compatible with permissionless challenging, as in BitVM3 [25]. This distinction affects the bridge’s security assumptions: the honest challenger needed to contest an invalid withdrawal need not belong to a set fixed during setup.

Organization. Section 2 gives a technical overview, and Section 3 recalls the required definitions and prior results. Section 4 defines the disclosure functions over groups, fields, and bits. Section 5 presents $\Pi_{\mathbb{G}}$, Section 6 proves the composition theorem required for projectivization, and Section 7 presents the projective scheme. Finally, in Appendix A, we discuss some optimizations that enable further reduction in garbled program size via complex multiplication on the BN254 elliptic curve.

2 Technical Overview

Our goal is to obtain a garbling scheme for Groth16 verification, where the garbling and evaluation algorithms are natively expressed over elliptic curves. In a little more detail, we wish to obtain a partial garbling scheme for the conditional disclosure function that takes as input a secret s from the garbler, a proof $\pi = (A, B, C)$ from the evaluator, and that discloses s to the evaluator iff the proof is invalid. To use this for two-party computation or trust-minimized Bitcoin bridges—our motivating application—the evaluator’s input needs to be encoded bit by bit. This means, we require a projective garbling scheme.

2.1 Background on Groth16 verification

A Groth16 proof is a triple (A, B, C) , where $A, C \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$. The source groups and the target group $\mathbb{G}_T$ have the same prime order r , and the pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is bilinear. We discuss, in more detail, the algebraic setting and the definition of these groups in Section 3. We write all three groups additively, so

$$e(aV, W) = a e(V, W) = e(V, aW)$$

for any $a \in \mathbb{F}_r$. With the verification key and public statement fixed, the Groth16 [14] verification equation can be written as

$$P(A, B, C) := e(A, B) + e(C, H_1) - D = 0,$$

for public constants $H_1 \in \mathbb{G}_2$ and $D \in \mathbb{G}_T$. Our goal is to construct a garbling scheme that reveals a garbler-sampled secret s when this check fails ($P \neq 0$) on an encoding of P and hides the secret when the check passes.

2.2 Group encodings for conditional disclosure of a secret

Observe that if we encode the evaluator’s input $\pi = (A, B, C)$ using (simple) elliptic curve operations, then the task of constructing a garbling scheme for Groth16 verification is much easier.

Consider an encoding of a point V of the form

$$\llbracket V \rrbracket = cV + K,$$

where the garbler chooses a secret bit $c \in \{0, 1\}$ and a masking point K . These are the basic group encodings underlying Argo MAC [11]. They support three operations that we use below.

First, encodings with the same coefficient can be added:

$$(cV_1 + K_1) + (cV_2 + K_2) = c(V_1 + V_2) + (K_1 + K_2).$$

Second, pairing against a fixed point preserves the coefficient:

$$e(cV + K, H) = c e(V, H) + e(K, H).$$

In both cases the garbler can compute the new mask $e(K, H)$ before learning the input. Third, a public linear combination of encodings of the same point combines their secret coefficients:

$$\sum_i 2^i (c_i V + K_i) = \left(\sum_i 2^i c_i \right) V + \sum_i 2^i K_i.$$

These operations require local group computations but no additional garbled material.

We wish to garble the following function:

$$f_{\text{cds}}^{\mathbb{G}}((A, B, C), s) = \begin{cases} \perp & \text{if } P = 0, \\ s & \text{if } P \neq 0, \end{cases}$$

where $s \in \{0, 1\}^\lambda$ is the garbler's private input, (A, B, C) is the evaluator's public input, and $P := P(A, B, C)$ is computed via the pairing check defined above. On an invalid proof, the function discloses the garbler's secret s to the evaluator. On a valid proof, the function outputs $\perp$ so by privacy, the evaluator should learn nothing about s .

Suppose, looking ahead, that we are able to design an evaluation procedure that outputs $Y = cP + K$ and that the garbler has published a commitment $h = \text{H}_{\text{com}}(K)$ to the mask K , as part of the garbled program. The evaluator knows P from the public proof. If $P \neq 0$, the two candidates for the mask Y and $Y - P$ are distinct, and one can learn c by comparing its hashes against the published hash of K . If $P = 0$, both candidates equal K , so this test reveals nothing about c .

We will eventually set things up such that recovering the value c is enough to disclose the secret (by deriving a one-time pad from it to encrypt s). But with only 2 candidates, an adversary can guess c directly, derive the mask at that guess and succeed with probability $1/2$. In other words, one bit is too little entropy to protect s . We therefore use a vector of κ independent bits $\vec{c} = (c_1, \dots, c_\kappa)$ and publish

$$\text{ct} = s \oplus \text{H}_{\text{key}}(\vec{c}).$$

We have reduced our task to arranging for an evaluator to obtain $Y_t = c_t P + K_t$ for every $t \in [\kappa]$. Assume she also holds $h_t = \text{H}_{\text{com}}(t, K_t)$. An invalid proof then reveals the entire coefficient vector and hence s . On a valid proof, the output values are independent of $\vec{c}$ and therefore s stays hidden under the one-time pad.

2.3 Garbling Groth16 verification with group encodings

We now describe how to build a vector encoding of P from vector encodings of A, B, C using the properties listed above. For convenience, we restate the equation for computing P :

$$P = e(A, B) + e(C, H_1) - D.$$

By linearity, we can describe how to handle each of these terms individually and then sum them up.

Encoding fixed pairing with C for free. Assume the evaluator holds the encoding vector

$$(c_t C + K_t^C)_{t \in [\kappa]}.$$

Pairing its value indexed by t against the fixed point H_1 gives

$$c_t e(C, H_1) + e(K_t^C, H_1),$$

which is an encoding of $e(C, H_1)$ under coefficient c_t .

Pairing inputs. For $e(A, B)$, the analogous computation gives

$$e(A, c_t B + K_t^B) = c_t e(A, B) + e(A, K_t^B).$$

The mask now depends on A . Because the garbler does not know A at setup, it cannot prepare the output commitment. Remember, for secret disclosure, a crucial property we used was that the garbler can compute and commit to the output mask during garbling. (The same difficulty arises if one instead encodes A and pairs it against a cleartext B .)

Rekeying gadget. We wish to construct a *rekeying gadget* that can be used to *correct* the input-dependent mask by one that the garbler can compute at garbling time. We first explain the correction for a single fixed index t , and then show how to share the expensive part across all $t \in [\kappa]$. For this, we first modify how the garbler samples the mask for the encoding of B . Let the corresponding mask be $K^B = kU$ for some randomly sampled base $U \in \mathbb{G}_2$ (which the garbler sends to the evaluator) and some private scalar $k \in \mathbb{F}_r$ known only to the garbler.

Our first step is to give the evaluator an “encoding” of A under the coefficient $-k \bmod r$. In more detail, we wish for the evaluator to obtain $Q = -kA + R$ for a random $R \in \mathbb{G}_1$ known to the garbler. Assume momentarily that this can be done efficiently. Pairing Q against the fixed base U gives

$$e(Q, U) = -k \cdot e(A, U) + e(R, U) = -e(A, K^B) + e(R, U),$$

and adding this to the original pairing cancels $e(A, K^B)$, leaving $c \cdot e(A, B) + e(R, U)$, with a mask $e(R, U)$ the garbler can commit to in advance. This addresses the above problem for one entry of the encoding vector at the cost of this special kind of encoding where the scalar k takes a value in $\mathbb{F}_r$.

We have so far restricted ourselves to simple encodings where the scalars take values in $\{0, 1\}$ so using an arbitrary scalar from $\mathbb{F}_r$ is not allowed by our encoding procedure. However, we can use the linear-combination property above to achieve this: write $-k \bmod r$ in binary i.e.,

$$-k \equiv \sum_{i \in [\log r]} 2^i b_i \pmod{r}$$

and encode A independently at $[\log r]$ entries with the binary digits $b_i \in \{0, 1\}$ as private coefficients and masks $M_i \in \mathbb{G}_1$ randomly sampled by the garbler. Now, the evaluator holds $b_i A + M_i$ for $i \in [\log r]$ and using the linear combination property, can obtain

$$\left(\sum_{i \in [\log r]} 2^i b_i \right) \cdot A + R = -kA + R$$

where the garbler defines

$$R := \sum_{i \in [\log r]} 2^i M_i$$

according to the same public linear combination.

Optimizing cost of rekeying gadget with correlated masks. As presented, correcting each element would require fresh instances of these simple encodings, leading to a total of $\kappa [\log r]$ simple encodings. In fact, the dominant cost of our protocol is *generating* each such encoding and therefore we wish to minimize this. We now describe an optimization that uses only one correction (i.e. $[\log r]$ simple encodings) to rekey the entire pairing output.

The key idea is for all indices to use the same secret scalar k , while the public base U_t varies with t . Concretely, the garbler samples independent $U_t \xleftarrow{\$} \mathbb{G}_2$ for $t \in [\kappa]$ and sets $K_t^B = kU_t$. The single point $Q = -kA + R$ then corrects all κ encoded values. For each $t \in [\kappa]$, adding

$$e(Q, U_t) = -e(A, K_t^B) + e(R, U_t)$$

cancels the input-dependent mask and leaves

$$c_t e(A, B) + e(R, U_t),$$

with a mask $e(R, U_t)$ the garbler can commit to in advance. We need to be careful here as sharing one scalar across κ encodings introduces a correlation between the masks. However, since k is private and the bases U_t are randomly sampled by the garbler, the joint distribution of these masks looks like a tuple of uniformly random group elements via decisional Diffie-Hellman in $\mathbb{G}_2$. We formalize this in Section 5.

Secret disclosure. Encode B and C with the same coefficient vector $\vec{c}$. The evaluator computes

$$Y_t = e(A, \llbracket B \rrbracket_t) + e(Q, U_t) + e(\llbracket C \rrbracket_t, H_1).$$

The garbler commits to

$$K_t = e(R, U_t) + e(K_t^C, H_1) + c_t D.$$

Substituting the encodings gives $Y_t = c_t P + K_t$. The candidate test from the previous subsection, applied to each entry $t \in [\kappa]$ now recovers $\vec{c}$ whenever $P \neq 0$.

2.4 Garbling the group encoding itself

The above garbling scheme assumed an evaluator can obtain group encodings of its input via some encoding procedure. For our application, we require that the garbling scheme is projective i.e., encodes inputs bitwise. To projectivize the scheme, we garble the group encoding function itself. In this new two-input function, the garbler holds the encoding information and the evaluator holds the point to be encoded. Therefore, the original problem now reduces to a partial garbling scheme for the group encoding function itself!

Groups to fields. This was the focus of prior works [11, 13] that use an information-theoretical partial garbling scheme (ITPG) to address the first step of this transformation. For a fixed masking point, elliptic-curve addition is expressed by low-degree polynomials in affine input coordinates (when the output is represented using Jacobian coordinates). Ishai–Wee partial garbling then produces an encoding vector that is affine over the base field [16, 11, 13, 17].

There is an additional subtlety that arises when *changing the domain* of the computation from groups to fields. A point in $\mathbb{G} \in \{\mathbb{G}_1, \mathbb{G}_2\}$ is by definition, on the corresponding elliptic curve. An arbitrary pair of coordinates $(u, v) \in \mathbb{F} \times \mathbb{F}$, where $\mathbb{F} \in \{\mathbb{F}_p, \mathbb{F}_{p^2}\}$, may not lie on the curve at all so the polynomial derived above would compute some tuple of points when evaluated on this pair, but there is no guarantee this corresponds to some encoding $cV + K$ for any point $V \in \mathbb{G}$. We must therefore treat an off-curve input (i.e. an invalid *representation* of an element in $\mathbb{G}$) the same way we treat a failing proof i.e., it should disclose s . Once again, we can do this via ITPGs. The curve equation $g(u, v) := v^2 - u^3 - \beta$ is itself a low degree polynomial in u, v , so we garble $sg(u, v)$ with the same information-theoretic scheme used for the addition law above, once for each of A , B , and C . On the curve, $g \equiv 0$, and the privacy of the underlying scheme at a fixed output of 0

hides s unconditionally. Off the curve, $g(u, v) \neq 0$, the garbled output is $sg(u, v)$, the evaluator can compute $g(u, v)$ in the clear by herself; she divides by the known nonzero $g(u, v)$ and recovers s directly. This is the same check described by [11, 17].

Fields to bits. The second transformation garbles the affine encoding $\vec{a}x + \vec{b}$ from the bits of x . Duty-Free Bits provides an efficient way to do this, accepting inputs from a $\lceil \log_2 p \rceil$ -bit domain, with bitstrings interpreted modulo p [17, Theorem 5.2].

As an additional contribution, we show how this technique can be applied, in a black box manner, to $\mathbb{F}_{p^2}$ as well. For an extension-field input $x = x_0 + \eta x_1$, we sample $\vec{T} \in \mathbb{F}_{p^2}^S$ and write

$$\vec{a}x + \vec{b} = (\vec{a}x_0 + \vec{T}) + ((\vec{a}\eta)x_1 + \vec{b} - \vec{T}).$$

Each summand, expressed in base-field components, is affine in one component of the input. The two summands form a uniform additive sharing of the desired encoding vector, so revealing the separately garbled outputs leaks nothing beyond their sum. We elaborate on this in Section 7.

2.5 Note on composition

The construction above moves through three input representations: group elements, field coordinates, and finally bits. We present this using a compositional approach. Suppose an *outer* garbler scheme's evaluator expects an encoding $\text{Enc}(\text{ek}, x)$. We view Enc itself as a two-input function: the garbler holds ek , while the evaluator holds x . Garbling this function with an *inner* scheme produces the encoding expected by the outer evaluator, while replacing the outer scheme's encoding interface by that of the inner scheme.

In our construction, this gives the sequence

$$\underbrace{\Pi_{\mathbb{G}}}_{\text{group-affine encoding}} \xrightarrow{\text{garble Enc}_{\mathbb{G}}} \underbrace{\Pi_{\mathbb{F}}}_{\text{field-affine encoding}} \xrightarrow{\text{garble Enc}_{\mathbb{F}}} \underbrace{\Pi_{\text{bin}}}_{\text{bit encoding}}$$

We believe this is the most natural way to present the final scheme: as a composition of several partial garbling schemes, each one progressively simplifying the input domain and encoding procedure.

Domain switching requires some additional care, such as checking the validity of a representation when reducing bits to a field element or field coordinates to a curve point, and we wish to handle these by one common interface. We define a composition procedure general enough to capture this and prove its security in Section 6. This simplifies both the presentation and the security proof of our headline result.

3 Preliminaries

In this section we recall results from prior work and additional preliminaries required to understand our construction. We first fix the algebraic setting, then provide brief background on elliptic curves and Groth16 verification, the syntax of partial garbling schemes, and information-theoretic partial garbling (ITPG) results of [16, 11, 13, 17].

3.1 Notation

We use λ to denote the computational security parameter and ρ for the statistical security parameter. We write $[m] = \{1, \dots, m\}$, use $z \xleftarrow{\$} Z$ for uniform sampling, and write $\approx$ for computational

indistinguishability. All groups are written additively, including the pairing target group. The identity of an elliptic-curve group is denoted $\mathcal{O}$, and additively written as 0 in some places.

3.2 Curve BN254

Our concrete instance uses the BN254 parameters specified for the pairing operation in EIP-197 [9], from the Barreto–Naehrig family [7]. Let

$$E : v^2 = u^3 + 3 \quad \text{over } \mathbb{F}_p, \quad \mathbb{G}_1 = E(\mathbb{F}_p).$$

The curve group has prime order r . Thus every affine pair satisfying this equation is in $\mathbb{G}_1$.

The second group we consider is a subgroup of $E(\mathbb{F}_{p^{12}})$, points satisfying E 's equation $v^2 = u^3 + 3$, but now with coordinates drawn from the much larger field $\mathbb{F}_{p^{12}}$ instead of $\mathbb{F}_p$. Computing directly with such coordinates would be expensive. Instead, $\mathbb{G}_2$ is *represented* by points on a smaller, related curve, over a much smaller field. This is known as a *twist*: a curve that becomes isomorphic to another once coordinates are allowed from a large enough field, even though no such isomorphism exists over the field its own coefficients live in. Concretely, the twist is

$$E' : v^2 = u^3 + 3/(9 + \eta) \quad \text{over } \mathbb{F}_{p^2} = \mathbb{F}_p[\eta]/(\eta^2 + 1),$$

and for $w \in \mathbb{F}_{p^{12}}$ satisfying $w^6 = 9 + \eta$, the map $(u, v) \mapsto (w^2u, w^3v)$ sends a point of E' to a point of $E(\mathbb{F}_{p^{12}})$, since

$$(w^3v)^2 = w^6v^2 = (9 + \eta) \left(u^3 + \frac{3}{9 + \eta} \right) = (9 + \eta)u^3 + 3 = (w^2u)^3 + 3,$$

and this map preserves the addition law. The point's representation on E' over $\mathbb{F}_{p^2}$ is used for efficiently computing underlying operations like addition, doubling, and scaling. We define $\mathbb{G}_2$ as the order- r subgroup of $E'(\mathbb{F}_{p^2})$. The group $E'(\mathbb{F}_{p^2})$ has order rh_2 for a cofactor $h_2 = 2p - r$ coprime to r , so $\mathbb{G}_2$ is its unique subgroup of order r . A pair of $\mathbb{F}_{p^2}$ coordinates satisfying the twist equation is therefore a point of $E'(\mathbb{F}_{p^2})$, but lies in $\mathbb{G}_2$ iff multiplication by r maps it to $\mathcal{O}$.

We use a nondegenerate type-3 pairing

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T, \quad e(aV, bW) = ab e(V, W),$$

where $\mathbb{G}_T$ is the order- r subgroup of $\mathbb{F}_{p^{12}}^\times$. Its multiplicative operation is written as addition throughout the paper.

Groth16 verification. Write the Groth16 [14] verification key as

$$\mathbf{vk} = (\alpha; \beta, \gamma, \delta; \mathbf{vk}_0, \dots, \mathbf{vk}_\ell),$$

with $\alpha, \mathbf{vk}_i \in \mathbb{G}_1$ and $\beta, \gamma, \delta \in \mathbb{G}_2$. For a statement $\xi = (\xi_1, \dots, \xi_\ell) \in \mathbb{F}_r^\ell$, set $X = \mathbf{vk}_0 + \sum_i \xi_i \mathbf{vk}_i$. Groth16 accepts a well-formed proof (A, B, C) when

$$e(A, B) = e(\alpha, \beta) + e(X, \gamma) + e(C, \delta).$$

We fix $\mathbf{vk}$ and ξ at garbling time and write

$$H_1 := -\delta, \quad D := e(\alpha, \beta) + e(X, \gamma), \quad P(A, B, C) := e(A, B) + e(C, H_1) - D.$$

The pairing check is then $P = 0$. Throughout this paper, we work in the setting where the statement is fixed ahead of time, and the evaluator's input is a Groth16 proof (A, B, C) .

3.3 Assumptions

We use a nondegenerate type-3 pairing where it is standard to assume DDH in $\mathbb{G}_2$: for a generator G and independent uniform $a, b, z \in \mathbb{F}_r$, the tuples (G, aG, bG, abG) and (G, aG, bG, zG) are computationally indistinguishable. We work in the random oracle model. In particular, for commitments we use a τ -bit random oracle H_{com} and to derive one-time pads we use an independent λ -bit domain H_{key} . The security of our final projective scheme will also use a circular correlation robust hash, the symmetric-key assumption behind the free-XOR technique [19], used by [17] which we use as a subprotocol. This assumption is trivially implied by a random oracle, but is weaker and can be instantiated more efficiently.

3.4 Partial garbling schemes

We use the definition of a partial garbling scheme from [17, Def. 1], which in turn follows Ishai and Wee [16] but separates the garbling of the program from the encoding of the input. The scheme is called *partial* because only the garbler’s part of the input is hidden; the evaluator’s part is public and is handed to the simulator.

Definition 1 (Partial garbling). A partial garbling scheme for $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$ consists of efficient algorithms (**Garb**, **Enc**, **Eval**):

1. **Garb**($1^\lambda, y$) outputs a garbled program $\tilde{f}$ and private encoding information ek .
2. **Enc**(ek, x) outputs an encoding $\tilde{x}$ of the evaluator’s input x .
3. **Eval**($\tilde{f}, x, \tilde{x}$) outputs a value in $\mathcal{Z}$.

For fixed x, y , let $(\tilde{f}, \text{ek}) \leftarrow \text{Garb}(1^\lambda, y)$ and $\tilde{x} \leftarrow \text{Enc}(\text{ek}, x)$. The scheme has correctness error δ if

$$\Pr[\text{Eval}(\tilde{f}, x, \tilde{x}) \neq f(x, y)] \leq \delta$$

for every fixed x, y . It is private if an efficient simulator $\mathcal{S}$ satisfies

$$(\tilde{f}, \tilde{x}) \approx \mathcal{S}(1^\lambda, x, f(x, y))$$

for every fixed x, y . Privacy is statistical when the distributions have negligible statistical distance, and perfect when they are identical.

Above, $\mathcal{X}$ is the evaluator’s input domain, $\mathcal{Y}$ the garbler’s private input domain and $\mathcal{Z}$ the output domain.

Affine and projective encodings. Suppose $\mathcal{X} \subseteq \prod_{i=1}^d M_i$, where the M_i are abelian groups. For an affine encoding, we write

$$\text{Enc}(\text{ek}, x) = \left((\sigma_{i,t}(x_i) + b_{i,t})_{t \in [S_i]} \right)_{i \in [d]}.$$

The vector indexed by $t \in [S_i]$ is the encoding vector associated with x_i , and S_i is its dimension. We extend [17]’s definition to capture our notion of group affine encodings. In particular, we allow “coefficients” of an affine encoding to be arbitrary endomorphisms of that coordinate’s group, rather than from a ring. Over a field, this vector has the familiar form

$$\vec{a}_i x_i + \vec{b}_i.$$

Over a curve group, the maps $\sigma_{i,t}$ are group endomorphisms. In our construction $\Pi_{\mathbb{G}}$, multiplication by $c = 0$ and $c = 1$ correspond to the zero and identity endomorphisms. In Appendix A, we consider a larger set of endomorphisms that enable certain optimizations.

An encoding of a bit string is *projective* if, for every input bit x_i , the corresponding part of the encoding is one of two fixed strings $E_{i,0}$ and $E_{i,1}$, namely E_{i,x_i} . Equivalently,

$$E_{i,x_i} = E_{i,0} + x_i(E_{i,1} - E_{i,0})$$

componentwise over $\mathbb{Z}_2$. This allows the encode procedure to be instantiated via 1-out-of-2 oblivious transfer or, in our motivating application, through one-time signatures.

3.5 ITPGs for group encodings

In this work, we present a new partial garbling scheme that has a simple encoding function which is, in the above sense, affine over groups. We use results from prior work [11, 13, 17] to efficiently projectivize our scheme by garbling this group encoding function. In this section, we present the necessary background. Recall from Section 2 that producing this group encoding and checking whether a claimed representation of a group element is valid, can both be expressed as a low-degree polynomial whose coefficients are garbler-private. We start by stating a lemma via Ishai and Wee (restated in the language of [17]) that describes an ITPG for this function.

Lemma 2 (Sum of monomials [16]). *Let $\mathcal{R}$ be a finite commutative ring and fix monomials $m_1, \dots, m_J$ of positive degrees $d_1, \dots, d_J$, with $J \geq 1$. For private coefficients $a_0, \dots, a_J \in \mathcal{R}$, the function*

$$h(x; a) = a_0 + \sum_{j=1}^J a_j m_j(x)$$

has a perfectly private affine partial garbling scheme. For each input variable x_i , define

$$S_i := \sum_{j=1}^J \deg_{x_i}(m_j).$$

The scheme has an empty garbled program, and the encoding vector associated with x_i has dimension S_i . Its total dimension is therefore

$$\sum_i S_i = \sum_{j=1}^J d_j.$$

Proof. We refer the reader to [16] for a proof. □

This is information-theoretic in the sense that it does not rely on any hardness assumptions, and there is no resulting garbled program. Rather, the private computation is now delegated to the encoding function which is affine over the ring $\mathcal{R}$, which in our case will be a field $\mathbb{F} \in \{\mathbb{F}_p, \mathbb{F}_{p^2}\}$.

Argo encodings. Recall from Section 2 that we encode a point $V \in \mathbb{G}$, for $\mathbb{G} \in \{\mathbb{G}_1, \mathbb{G}_2\}$, as $cV + K$, where $c \in \{0, 1\}$ and $K \in \mathbb{G}$ is a uniform mask, which we sometimes refer to as the garbler's key.

Theorem 3 (Argo encodings from coordinates [11]). *The function $(V, (c, K)) \mapsto cV + K$, where V is a point of E or of E' other than $\mathcal{O}$, given by its affine coordinates, and the garbler’s input is $c \in \{0, 1\}$ and K a randomly sampled point on the same curve, has a statistical affine partial garbling scheme, with correctness error $O(1/r)$, in which the encoding vectors associated with u and v have dimensions 7 and 5, respectively. The encoding function is affine over $\mathbb{F}_p$ for E and over $\mathbb{F}_{p^2}$ for E' .*

Proof. This result is a special case of a more general result due to [11]. We elaborate on this special case for completeness. We start by computing Jacobian coordinates of $cV + K$ above, using the group addition law for the curve $E_\beta : v^2 = u^3 + \beta$, written as polynomials. For $c = 1$ and $K = (a, b)$,

$$\begin{aligned} X &= au^2 + a^2u - 2bv + 2\beta, \\ Y &= 3abu^2 - 3a^2uv + bv^2 - (a^3 + 4\beta)v + 3b\beta, \\ Z &= u - a. \end{aligned} \tag{1}$$

If $c = 0$, instead use a constant Jacobian representative of K : $(a, b, 1)$ or $(1, 1, 0)$ for $K = \mathcal{O}$. If $c = 1$ and $K = \mathcal{O}$, use $(u, v, 1)$. These addition formulas are incomplete when $c = 1$ and $K = V$. Substituting $(a, b) = (u, v)$ and using $v^2 = u^3 + \beta$ gives $(X, Y, Z) = (0, 0, 0)$. However, for a uniformly sampled mask K , this occurs with probability $1/r$. In Appendix A, we discuss an alternate approach to remedying this which is slightly more costly but works in the general case. To randomize the Jacobian representation of the output a fresh mask $\mu \in \mathbb{F}^\times$ is sampled and we use Lemma 2 to compute the polynomial outputs $(\mu^2 X, \mu^3 Y, \mu Z)$, a randomized Jacobian coordinate representation. Below, we tabulate the degree of the associated polynomials and how it influences the size of the resulting affine encodings, counted according to Lemma 2.

Output	Positive-degree monomials	u -dimension	v -dimension
X	u^2, u, v	3	1
Y	u^2, uv, v^2, v	3	4
Z	u	1	0

Summing these contributions, the encoding vectors associated with u and v have dimensions 7 and 5, respectively, for total dimension 12. $\square$

Validity checks. Finally, we will need to check whether a claimed representation of a group element is well formed, and disclose the garbler’s secret when it is not. Once again, we can use ITPGs for low-degree polynomials. Given a public polynomial $g(u, v) = v^2 - u^3 - \beta$ vanishing exactly on well-formed inputs, and a secret s the garbler wants hidden on well-formed inputs and disclosed otherwise, Lemma 2 garbles $sg(u, v)$ with encoding vectors of dimensions 3 and 2 for u and v respectively. On a well-formed input, $g(u, v) = 0$, and privacy of Lemma 2 at a fixed output of 0 hides s . On an invalid representation, the evaluator obtains $sg(u, v)$ for $g(u, v) \neq 0$ which she can locally compute from public (u, v) , and divide to recover s . This is the mechanism described in [11, 17].

4 Disclosure Functions across Input Domains

The function being garbled is most naturally expressed over groups. However, we eventually want a projective garbling scheme i.e., one that accepts inputs bitwise. Therefore, we compile the function

over groups to one over fields (affine coordinates of curve points), and finally to one over the bit decomposition of affine coordinates.

As we switch the input domain of the function being garbled, we need to include input validation checks. For example, we need to ensure that field coordinates correspond to valid elliptic curve points. We extend the disclosure function to reveal the garbler's secret on any such invalid input.

Definition 4 (Conditional disclosure). For a predicate **Valid** on a domain $\mathcal{X}$, its disclosure function is

$$f : \mathcal{X} \times \{0, 1\}^\lambda \longrightarrow \{0, 1\}^\lambda \cup \{\perp\}, \quad f(x, s) = \begin{cases} \perp, & \text{Valid}(x), \\ s, & \text{otherwise.} \end{cases}$$

Group inputs. Let

$$\mathcal{X}_{\mathbb{G}} = \mathbb{G}_1 \times E'(\mathbb{F}_{p^2}) \times \mathbb{G}_1, \quad x = (A, B, C).$$

We let B range over the twist group rather than over $\mathbb{G}_2$. Looking ahead, this is because the garbling scheme will receive B as coordinates and cannot know in advance whether the point they represent has order r ; we include that subgroup check as part of the validity function. Define

$$\text{Valid}_{\mathbb{G}}(A, B, C) = \begin{cases} [P(A, B, C) = 0], & rB = \mathcal{O}, \\ \text{false}, & rB \neq \mathcal{O}. \end{cases}$$

Let $f_{\text{cds}}^{\mathbb{G}}$ be the disclosure function of **Valid** $_{\mathbb{G}}$.

Field inputs. Let

$$\mathcal{X}_{\mathbb{F}} = \mathbb{F}_p^2 \times \mathbb{F}_{p^2}^2 \times \mathbb{F}_p^2, \quad x = (A_u, A_v; B_u, B_v; C_u, C_v).$$

The map $\iota_{\mathbb{F}} : \mathcal{X}_{\mathbb{F}} \rightarrow \mathcal{X}_{\mathbb{G}} \cup \{\text{invalid}\}$ interprets the three coordinate pairs as affine points when they satisfy their respective curve equations, and returns the symbol **invalid** otherwise. Define

$$\text{Valid}_{\mathbb{F}}(x) = [\iota_{\mathbb{F}}(x) \neq \text{invalid}] \wedge \text{Valid}_{\mathbb{G}}(\iota_{\mathbb{F}}(x)).$$

Its disclosure function is $f_{\text{cds}}^{\mathbb{F}}$. On the domain $\mathcal{X}_{\mathbb{F}}^{\text{on}} = \{x \in \mathcal{X}_{\mathbb{F}} : \iota_{\mathbb{F}}(x) \neq \text{invalid}\}$, we have

$$f_{\text{cds}}^{\mathbb{F}}(x, s) = f_{\text{cds}}^{\mathbb{G}}(\iota_{\mathbb{F}}(x), s).$$

Outside this domain, $f_{\text{cds}}^{\mathbb{F}}(x, s) = s$. The secret is disclosed if any of the curve check, subgroup check, or pairing check fail.

Bit inputs. Set $w = \lceil \log_2 p \rceil$ and let $\mathcal{X}_{\text{bin}} = \{0, 1\}^{8w}$. A string is divided into eight blocks, in the order

$$(A_u, A_v, B_{u,0}, B_{u,1}, B_{v,0}, B_{v,1}, C_u, C_v),$$

where $B_u = B_{u,0} + \eta B_{u,1}$ and similarly for B_v . The function reads a block $(z_0, \dots, z_{w-1})$ as the integer $\sum_{j=0}^{w-1} 2^j z_j$, then reduces it modulo p . Let $\iota_{\text{bin}} : \mathcal{X}_{\text{bin}} \rightarrow \mathcal{X}_{\mathbb{F}}$ be the resulting map, and define

$$\text{Valid}_{\text{bin}} = \text{Valid}_{\mathbb{F}} \circ \iota_{\text{bin}}, \quad f_{\text{cds}}^{\text{bin}}(\bar{x}, s) = f_{\text{cds}}^{\mathbb{F}}(\iota_{\text{bin}}(\bar{x}), s).$$

This is the **bin-to- $\mathbb{F}_p$** map used by Duty-Free Bits [17, Section 2 and Theorem 5.2].

5 PGS for $f_{\text{cds}}^{\mathbb{G}}$ with affine encoding over groups

In this section we construct $\Pi_{\mathbb{G}}$, a partial garbling scheme for $f_{\text{cds}}^{\mathbb{G}}$ with affine group encodings. Set

$$n = \lceil \log_2 r \rceil, \quad \kappa = \lambda.$$

The encoding vector for A has dimension n , while those for B and C each have dimension κ . Privacy and correctness of this scheme is independent of how elliptic curve points are represented as group elements.

For $z \in \mathbb{F}_r$, write its binary decomposition as

$$z = \sum_{i=0}^{n-1} 2^i b_i \quad \text{in } \mathbb{F}_r, \quad b_i \in \{0, 1\}.$$

5.1 Construction

Parties agree on public parameters including the Groth16 verification key, fixed statement, and a unique session identifier sid . The vectors $\llbracket A \rrbracket$, $\llbracket B \rrbracket$, and $\llbracket C \rrbracket$ denote the encodings of the three proof points. The point M_i masks $\llbracket A \rrbracket_i$, while K_t^B and K_t^C mask $\llbracket B \rrbracket_t$ and $\llbracket C \rrbracket_t$, respectively. The value K_t is the resulting target-group mask committed to in the garbled program. Formally, $\Pi_{\mathbb{G}}$ is defined as follows.

$\Pi_{\mathbb{G}}$: disclosure with affine group encodings
Garb$_{\mathbb{G}}$($1^\lambda, s$). Sample independently $c_t \xleftarrow{\$} \{0, 1\}, \quad U_t \xleftarrow{\$} \mathbb{G}_2, \quad K_t^C \xleftarrow{\$} \mathbb{G}_1 \quad (t \in [\kappa]), \quad k \xleftarrow{\$} \mathbb{F}_r, \quad M_i \xleftarrow{\$} \mathbb{G}_1 \quad (0 \leq i < n).$ Expand $-k = \sum_i 2^i b_i$ with $b_i \in \{0, 1\}$, and set $R = \sum_i 2^i M_i, \quad K_t^B = kU_t, \quad t \in [\kappa].$ For each $t \in [\kappa]$, compute $K_t = e(R, U_t) + e(K_t^C, H_1) + c_t D, \quad h_t = \text{H}_{\text{com}}(\text{sid}, t, K_t).$ Set $\text{ct} = s \oplus \text{H}_{\text{key}}(\text{sid}, \vec{c})$ and output $\begin{aligned} \tilde{f}_{\mathbb{G}} &= ((U_t)_t, (h_t)_t, \text{ct}), \\ \text{ek}_{\mathbb{G}} &= ((b_i)_i, (M_i)_i, (c_t)_t, (K_t^B)_t, (K_t^C)_t). \end{aligned}$ Enc$_{\mathbb{G}}$($\text{ek}_{\mathbb{G}}, (A, B, C)$). Output the three vectors $\begin{aligned} \llbracket A \rrbracket_i &= b_i A + M_i & (0 \leq i < n), \\ \llbracket B \rrbracket_t &= c_t B + K_t^B & (t \in [\kappa]), \\ \llbracket C \rrbracket_t &= c_t C + K_t^C & (t \in [\kappa]). \end{aligned}$

$\Pi_{\mathbb{G}}$: disclosure with affine group encodings (continued)

$\text{Eval}_{\mathbb{G}}(\tilde{f}_{\mathbb{G}}, (A, B, C), ([A], [B], [C]))$.

1. If $rB \neq \mathcal{O}$, find at each $t \in [\kappa]$ the unique $d_t \in \{0, 1\}$ such that

$$d_t(rB) = r[B]_t.$$

If any t has no unique solution, return $\perp$. Otherwise return $\text{ct} \oplus H_{\text{key}}(\text{sid}, (d_t)_t)$.

2. If $rB = \mathcal{O}$, compute $P = P(A, B, C)$. If $P = 0$, return $\perp$. Otherwise check that each $[B]_t \in \mathbb{G}_2$, and return $\perp$ if a check fails. Set

$$Q = \sum_i 2^i [A]_i, \quad Y_t = e(A, [B]_t) + e(Q, U_t) + e([C]_t, H_1).$$

Find at each $t \in [\kappa]$ the unique $d_t \in \{0, 1\}$ satisfying

$$H_{\text{com}}(\text{sid}, t, Y_t - d_t P) = h_t.$$

If any t has no unique solution, return $\perp$. Otherwise return $\text{ct} \oplus H_{\text{key}}(\text{sid}, (d_t)_t)$.

The encoding is affine in the sense of Definition 1. The garbled program has κ elements of $\mathbb{G}_2$, κ hashes of τ bits, and a λ -bit ciphertext.

Lemma 5 (Correctness of $\Pi_{\mathbb{G}}$). *For every fixed $x \in \mathcal{X}_{\mathbb{G}}$ and $s \in \{0, 1\}^{\lambda}$, $\Pi_{\mathbb{G}}$ evaluates to $f_{\text{cds}}^{\mathbb{G}}(x, s)$ except with probability at most*

$$\delta_{\mathbb{G}} = \kappa 2^{-\tau}.$$

Proof. Suppose first that $rB \neq \mathcal{O}$. Multiplication by r maps $K_t^B \in \mathbb{G}_2$ to $\mathcal{O}$, leaving

$$r[B]_t = c_t(rB).$$

Because $rB \neq \mathcal{O}$, we can recover $\vec{c}$ and return s .

Now suppose that $B \in \mathbb{G}_2$. Then

$$Q = \sum_i 2^i (b_i A + M_i) = -kA + R.$$

Bilinearity gives

$$\begin{aligned} e(A, [B]_t) + e(Q, U_t) &= c_t e(A, B) + k e(A, U_t) - k e(A, U_t) + e(R, U_t) \\ &= c_t e(A, B) + e(R, U_t). \end{aligned}$$

Adding the C term gives

$$Y_t = c_t (e(A, B) + e(C, H_1)) + e(R, U_t) + e(K_t^C, H_1) = c_t P + K_t.$$

If $P = 0$, the input is valid and the evaluator returns $\perp$. If $P \neq 0$, the two points $Y_t - dP$, for $d \in \{0, 1\}$, are distinct. The correct value $d = c_t$ hashes to h_t . Since different values collide with probability $2^{-\tau}$, we can use a union bound over the κ incorrect candidates which gives $\delta_{\mathbb{G}}$. Other than the negligible event of such a collision, the evaluator recovers $\vec{c}$ and therefore s . $\square$

For efficiency, our scheme uses correlated masks for the Argo encoding of B . This is secure via DDH in $\mathbb{G}_2$, and the privacy of our scheme uses the following lemma, which roughly states that masks sharing a scalar across independent bases look like independent masks.

Lemma 6 (DDH across independent bases). *For uniform $k \in \mathbb{F}_r$ and independent uniform $U_1, \dots, U_\kappa \in \mathbb{G}_2$, the distributions*

$$(U_t, kU_t)_{t \in [\kappa]} \quad \text{and} \quad (U_t, S_t)_{t \in [\kappa]}$$

are computationally indistinguishable, where the S_t are independent uniform points. The distinguishing advantage is at most $\kappa \text{Adv}_{\mathbb{G}_2}^{\text{DDH}}$.

Proof. This follows from a standard hybrid argument. $\square$

Note that for privacy when garbling a disclosure function, a simulator only needs to handle the case of valid inputs, as the other case is trivial to simulate. We state this below.

Lemma 7 (Valid inputs suffice). *A partial garbling scheme for the disclosure function of **Valid** is private if it has a simulator on every valid input.*

Proof. On an invalid input, the simulator receives the secret s as the function output. It can run the real garbling and encoding algorithms, reproducing the view exactly. On a valid input, use the simulator from the hypothesis. $\square$

Theorem 8 (Privacy of $\Pi_{\mathbb{G}}$). *Under DDH in $\mathbb{G}_2$ and in the random oracle model for $H_{\text{com}}, H_{\text{key}}$, $\Pi_{\mathbb{G}}$ is private. For a distinguisher making at most q queries to the oracle, its advantage is at most*

$$\varepsilon_{\mathbb{G}} = 2\kappa \text{Adv}_{\mathbb{G}_2}^{\text{DDH}} + 2q2^{-\kappa}.$$

Proof. By Lemma 7, fix a valid input $x = (A, B, C)$. Then $B \in \mathbb{G}_2$ and $P = 0$. On a valid input, a simulator samples $s^* \xleftarrow{\$} \{0, 1\}^\lambda$ and runs the real garbling and encoding algorithms with s^* in place of s . We start by considering the real view under s and show, via a hybrid argument, this is indistinguishable from a distribution independent of s .

1. In this hybrid, we replace K_t^B with uniform masks. For every value of k , the encoded points $\llbracket A \rrbracket_i = b_i A + M_i$ are independent uniform points. We may sample these encodings first, compute $Q = \sum_i 2^i \llbracket A \rrbracket_i$, and set $R = Q + kA$. This preserves the joint distribution exactly, including its correlation with k . With this sampling order,

$$K_t = e(Q, U_t) + e(A, kU_t) + e(K_t^C, H_1) + c_t D.$$

Now, k appears in the view only through (U_t, kU_t) . Lemma 6 allows us to replace kU_t by independent $S_t \xleftarrow{\$} \mathbb{G}_2$ with distinguishing advantage at most $\kappa \text{Adv}_{\mathbb{G}_2}^{\text{DDH}}$.

2. In this hybrid, we remove $\vec{c}$ from the definition of the output mask by relabeling. Set $L_t^B = c_t B + S_t$ and $L_t^C = c_t C + K_t^C$. These are uniform points, independent of $\vec{c}$ as S_t, K_t^C act as one-time masks. Substitution gives

$$\begin{aligned} K_t &= e(Q, U_t) + e(A, L_t^B - c_t B) + e(L_t^C - c_t C, H_1) + c_t D \\ &= e(Q, U_t) + e(A, L_t^B) + e(L_t^C, H_1) - c_t P \\ &= e(Q, U_t) + e(A, L_t^B) + e(L_t^C, H_1). \end{aligned}$$

Thus the bases, input encodings, committed masks, and commitment hashes can all be sampled without $\vec{c}$. Only the one-time padded ciphertext still depends on $\vec{c}$.

3. In this hybrid, we replace ct by a uniform λ -bit string. In the random-oracle model, these views agree unless the distinguisher queries $\mathbf{H}_{\text{key}}$ at $(\text{sid}, \vec{c})$. This vector is uniform in $\{0, 1\}^\kappa$ in the previous view, so q guesses hit it with probability at most $q2^{-\kappa}$.

The final distribution is independent of s . The same argument can be used to transform the simulator's distribution under s^* to this final distribution. Adding both errors give us the desired bound. $\square$

6 Composing Partial Garbling Schemes

In this section, we describe our approach for composing partial garbling schemes. A partial garbling scheme reduces private computation of $f(x, y)$ to private computation of its encoding function $\text{Enc}(\text{ek}, x)$, followed by public evaluation. The encoding function is itself a two-input function: x belongs to the evaluator and ek to the garbler. Garbling this function changes the encoding interface of the original scheme. This is the transformation we will use to projectivize $\Pi_{\mathbb{G}}$ by moving from group encodings to field encodings and then to bit encodings.

Theorem 9 (Composition). *Let $\Pi_f = (\text{Garb}_f, \text{Enc}_f, \text{Eval}_f)$ garble $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$, with deterministic encoding given ek_f . Let $\iota : \mathcal{X}' \rightarrow \mathcal{X}$ be a public domain translation map, and let $\Pi_g = (\text{Garb}_g, \text{Enc}_g, \text{Eval}_g)$ garble*

$$g(x', \text{ek}_f) = \text{Enc}_f(\text{ek}_f, \iota(x')).$$

Define $\Pi_f \circ_\iota \Pi_g$ as follows:

$$\begin{aligned} \text{Garb}(1^\lambda, y) : & \quad (\tilde{f}, \text{ek}_f) \leftarrow \text{Garb}_f(1^\lambda, y), \\ & \quad (\tilde{g}, \text{ek}_g) \leftarrow \text{Garb}_g(1^\lambda, \text{ek}_f), \\ & \quad \text{output } ((\tilde{f}, \tilde{g}), \text{ek}_g); \\ \text{Enc}(\text{ek}_g, x') := & \quad \text{Enc}_g(\text{ek}_g, x'); \\ \text{Eval}((\tilde{f}, \tilde{g}), x', \tilde{x}) := & \quad \text{Eval}_f(\tilde{f}, \iota(x'), \text{Eval}_g(\tilde{g}, x', \tilde{x})). \end{aligned}$$

The composed scheme garbles the function $(x', y) \mapsto f(\iota(x'), y)$. The resulting program size, correctness error, and privacy loss is the sum of the two schemes' respective sizes and errors. Its encoding is that of Π_g , and therefore inherits affineness or projectivity from Π_g .

Proof. Fix x', y . Unless inner evaluation fails, it returns $\text{Enc}_f(\text{ek}_f, \iota(x'))$, which is the input encoding expected by the outer evaluator. Unless outer evaluation fails, evaluation therefore returns $f(\iota(x'), y)$. A union bound gives the correctness claim.

For privacy, suppress the common security parameter and define the simulator by

$$\begin{aligned} (\tilde{f}, \tilde{x}_f) & \leftarrow \mathcal{S}_f(\iota(x'), z), \\ (\tilde{g}, \tilde{x}_g) & \leftarrow \mathcal{S}_g(x', \tilde{x}_f), \end{aligned}$$

returning $((\tilde{f}, \tilde{g}), \tilde{x}_g)$. Starting from the real view, first replace the inner garbled program and its input encoding by

$$\mathcal{S}_g(x', \text{Enc}_f(\text{ek}_f, \iota(x'))).$$

The resulting view is an efficient randomized function of

$$(\tilde{f}, \text{Enc}_f(\text{ek}_f, \iota(x'))).$$

By outer privacy, we can replace this pair by $\mathcal{S}_f(\iota(x'), z)$, which gives us the required simulator. The program and encoding claims follow from the construction. $\square$

Note that this composition requires a domain translation map. The bit-to-field map is well-defined on the entire domain because it reduces each integer modulo p . The field-to-group map is defined as a point translation only on $\mathcal{X}_{\mathbb{F}}^{\text{on}}$. To extend the disclosure function to the rest of $\mathcal{X}_{\mathbb{F}}$, we add the polynomial curve checks from Section 3. These checks reveal s exactly where a coordinate pair fails to define a point. The theorem can be applied repeatedly, each application changing the domain in which the evaluator's input is encoded.

7 Projectivizing $\Pi_{\mathbb{G}}$

Given the above recipe for composing partial garbling scheme, we now show how to projectivize our Groth16 garbling scheme. $\Pi_{\mathbb{G}}$ has an encoding that is affine over groups. To obtain a projective garbling scheme we apply Theorem 9 twice: once to garble $\text{Enc}_{\mathbb{G}}$ from the field coordinates of the proof, which gives a scheme $\Pi_{\mathbb{F}}$ with a field-affine encoding, and once more to garble that encoding from bits, which gives Π_{bin} , a projective scheme. For the final step, we introduce Lemma 12, which extends the projectivization of [17] from $\mathbb{F}_p$ to $\mathbb{F}_{p^2}$, for compatibility with point $B \in \mathbb{G}_2$.

7.1 From groups to fields

The garbling algorithm Garb_1 receives $\text{ek}_{\mathbb{G}}$ and instantiates the polynomial ITPG of Section 3 for every encoded point $\llbracket A \rrbracket_i$, $\llbracket B \rrbracket_t$, and $\llbracket C \rrbracket_t$. The algorithm Enc_1 outputs the corresponding field encodings, grouped by input coordinate, and Eval_1 evaluates the ITPGs and decodes the resulting Jacobian representatives.

For A , the encoding vectors associated with its two affine coordinates have dimensions $7n$ and $5n$. For each of B and C , the corresponding dimensions are 7κ and 5κ .

Privacy follows straightforwardly by the composition theorem.

Curve checks for input validity. Let g_A, g_B, g_C be the outputs of the curve check equations, with constants 3, $3/(9 + \eta)$, and 3, evaluated on points A, B and C respectively. Embed $\{0, 1\}^\lambda$ injectively into $\mathbb{F}_p$, requiring $2^\lambda \leq p$, and hence into $\mathbb{F}_{p^2}$. Let Π_γ concatenate independent instances of Lemma 2 for sg_A , sg_B , and sg_C . Each curve check contributes dimension 3 to the encoding of its first coordinate and dimension 2 to that of its second coordinate.

$\Pi_{\mathbb{F}}$: disclosure with affine field encodings

$\text{Garb}_{\mathbb{F}}(1^\lambda, s)$. Run

$$\begin{aligned} (\tilde{f}_{\mathbb{G}}, \text{ek}_{\mathbb{G}}) &\leftarrow \text{Garb}_{\mathbb{G}}(1^\lambda, s), \\ (\emptyset, \text{ek}_1) &\leftarrow \text{Garb}_1(1^\lambda, \text{ek}_{\mathbb{G}}), \\ (\emptyset, \text{ek}_\gamma) &\leftarrow \text{Garb}_\gamma(1^\lambda, s), \end{aligned}$$

with independent randomness. Output $\tilde{f}_{\mathbb{F}} = \tilde{f}_{\mathbb{G}}$ and $\text{ek}_{\mathbb{F}} = (\text{ek}_1, \text{ek}_\gamma)$, concatenated by coordinate.

$\text{Enc}_{\mathbb{F}}(\text{ek}_{\mathbb{F}}, x)$. Output $\tilde{x} = (\tilde{x}_1, \tilde{x}_\gamma)$, where $\tilde{x}_1 = \text{Enc}_1(\text{ek}_1, x)$ and $\tilde{x}_\gamma = \text{Enc}_\gamma(\text{ek}_\gamma, x)$.

$\text{Eval}_{\mathbb{F}}(\tilde{f}_{\mathbb{F}}, x, (\tilde{x}_1, \tilde{x}_\gamma))$. Compute $g_A(x), g_B(x), g_C(x)$ in the clear.

1. If any $g_V(x)$ is nonzero, compute $z_V = \text{Eval}_\gamma(\emptyset, x, \tilde{x}_\gamma)_V$ and return $z_V/g_V(x)$ as a λ -bit secret.

$\Pi_{\mathbb{F}}$: disclosure with affine field encodings (continued)

2. Otherwise compute $\tilde{x}_{\mathbb{G}} = \text{Eval}_1(\emptyset, x, \tilde{x}_1)$. If this fails, return $\perp$. Otherwise return

$$\text{Eval}_{\mathbb{G}}(\tilde{f}_{\mathbb{G}}, \iota_{\mathbb{F}}(x), \tilde{x}_{\mathbb{G}}).$$

Write $S_A^{\mathbb{F}}$ for the total dimension of the field encoding associated with the two coordinates of A , and define $S_B^{\mathbb{F}}$ and $S_C^{\mathbb{F}}$ analogously.

Theorem 10 (Correctness and Privacy of $\Pi_{\mathbb{F}}$). *Using the ITPGs for group encodings of Section 3 and under the assumptions of Theorem 8, $\Pi_{\mathbb{F}}$ is a private affine partial garbling scheme for $f_{\text{cds}}^{\mathbb{F}}$. Its garbled program is $\tilde{f}_{\mathbb{G}}$ and has field encodings of size*

$$S_A^{\mathbb{F}} = 12n + 5, \quad S_B^{\mathbb{F}} = 12\kappa + 5, \quad S_C^{\mathbb{F}} = 12\kappa + 5.$$

The encodings associated with A and C take values in $\mathbb{F}_p$, while the encoding associated with B takes values in $\mathbb{F}_{p^2}$.

Proof. If a coordinate pair is off its curve, the corresponding check returns $sg_V(x)$ exactly. Since $g_V(x) \neq 0$, this recovers s . If all pairs are on their curves, an evaluator obtains the correct group encodings exactly, and correctness then follows via Lemma 5.

For privacy, Lemma 7 allows us to fix a valid field input x . In particular, all three curve check outputs are zero, and $\iota_{\mathbb{F}}(x)$ is a valid group input. By the privacy of Lemma 2, the curve-check encodings can be replaced by independent simulations given x and their respective outputs i.e., 0. The remaining view is obtained by applying $\mathcal{S}_1$ to x and the outer group encoding. Theorem 8 replaces this outer view by its simulator on $(\iota_{\mathbb{F}}(x), \perp)$. Therefore, applying $\mathcal{S}_1$ to the simulated encodings and appending the independently simulated encodings for the curve checks gives us the full simulator. $\square$

7.2 From fields to bits

The last step is projectivization of field encodings via [17]. To obtain projective input encodings, we garble the affine functions defining the field encoding. For $w = \lceil \log_2 p \rceil$ and a bit block $\bar{x} = (x_0, \dots, x_{w-1}) \in \{0, 1\}^w$, write

$$\text{val}_p(\bar{x}) = \sum_{j=0}^{w-1} 2^j x_j \bmod p.$$

Theorem 11 (Prime-field projectivization [17, Theorem 5.2]). *Let $w = \lceil \log_2 p \rceil$, $S \geq 1$, and let the statistical parameter ρ be such that $\rho < w$. Assuming a circular-correlated hash function, as defined in [17], the map*

$$(\bar{x}, (\vec{a}, \vec{b})) \mapsto \vec{a} \text{val}_p(\bar{x}) + \vec{b}, \quad \vec{a}, \vec{b} \in \mathbb{F}_p^S,$$

has a projective partial garbling scheme with program size $O(w(\lambda + S))$ bits.

Theorem 11 provides an efficient scheme for coordinates taking values in $\mathbb{F}_p$. The coordinates of B , however, lie in $\mathbb{F}_{p^2}$, and an $\mathbb{F}_{p^2}$ -affine encoding is affine in the pair of $\mathbb{F}_p$ components of B_u but not in each component separately, so Theorem 11 does not apply to it directly. We resolve this via a random additive share.

Lemma 12 (Affine encodings over $\mathbb{F}_{p^2}$). *Let $w = \lceil \log_2 p \rceil$, $S \geq 1$. For $\bar{x} = (\bar{x}_0, \bar{x}_1) \in \{0, 1\}^{2w}$, define $x = \text{val}_p(\bar{x}_0) + \eta \text{val}_p(\bar{x}_1)$. The map*

$$(\bar{x}, (\vec{a}, \vec{b})) \mapsto \vec{a}x + \vec{b}, \quad \vec{a}, \vec{b} \in \mathbb{F}_{p^2}^S,$$

has a projective partial garbling scheme with program size $O(w(\lambda + S))$ bits.

Proof. Sample $\vec{T} \xleftarrow{\$} \mathbb{F}_{p^2}^S$ and write

$$\vec{a}(x_0 + \eta x_1) + \vec{b} = (\vec{a}x_0 + \vec{T}) + ((\vec{a}\eta)x_1 + \vec{b} - \vec{T}).$$

Writing $\vec{a} = \vec{a}_0 + \eta \vec{a}_1$ and likewise for the other vectors, the two summands have base-field components

$$\begin{aligned} U &= (\vec{a}_0x_0 + \vec{T}_0, \vec{a}_1x_0 + \vec{T}_1), \\ V &= (-\vec{a}_1x_1 + \vec{b}_0 - \vec{T}_0, \vec{a}_0x_1 + \vec{b}_1 - \vec{T}_1). \end{aligned}$$

Each is an affine function of one prime-field input of dimension $2S$. Apply Theorem 11 independently to the two maps. The evaluator adds the outputs component-wise to obtain the desired extension-field encodings.

A simulator given the output $U + V$ samples U uniformly, sets V to the difference, and invokes the two prime-field simulators. In total, we invoke two $2S$ -dimension instances over $\mathbb{F}_p$, resulting in the claimed size. $\square$

Let Π_2 denote the protocol that invokes instances of Theorem 11 and Lemma 12 as required above for each coordinate of A, B, C and concatenates. Its private inputs are the coefficients and offsets defining the corresponding affine encoding vectors in $\text{ek}_{\mathbb{F}}$. The resulting field encoding is then supplied to $\Pi_{\mathbb{F}}$.

We denote using

$$N_{\text{base}} = S_A^{\mathbb{F}} + 4S_B^{\mathbb{F}} + S_C^{\mathbb{F}}$$

the aggregate output dimension of the eight prime-field instances of Theorem 11.

Theorem 13 (Main). *Under DDH in $\mathbb{G}_2$, and in the random oracle model,*

$$\Pi_{\text{bin}} := \Pi_{\mathbb{F}} \circ \iota_{\text{bin}} \Pi_2$$

is a projective partial garbling scheme for $f_{\text{cds}}^{\text{bin}}$. Its program size is

$$|\tilde{f}_{\mathbb{G}}| + O(w(\lambda + N_{\text{base}})) \quad \text{bits.}$$

Proof. The bit-to-field map is well-defined and Theorem 9 applies to Theorem 10 and Π_2 . The inner encoding is projective: by Theorem 11 and Lemma 12, every input bit selects one of two fixed encodings in each instance. There are four direct prime-field instances and four more from splitting the two extension-field coordinates. Their aggregate output dimension sums to $S_A^{\mathbb{F}} + 4S_B^{\mathbb{F}} + S_C^{\mathbb{F}}$, as required. $\square$

Remark 14. In all instantiations, we will set parameters $\tau, \kappa, n, w = O(\lambda)$. Therefore, the resulting size of the garbled program for the projectivized scheme is $O(\lambda^2)$ bits. Note that the garbled program for the scheme over groups $\tilde{f}_{\mathbb{G}}$ itself consists of $\kappa = O(\lambda)$ commitments, each of size $\tau = O(\lambda)$ bits, resulting in $O(\lambda^2)$ size. Amazingly, this means that due to prior work [11, 13, 17], projectivization is asymptotically free! As discussed in Section 8, projectivization still concretely dominates the size of the garbled program.

8 Implementation and Concrete Costs

We implemented our main protocol Π_{bin} , a projective partial garbling scheme for the Groth16 disclosure function, in Rust. An anonymized version of our implementation is available at <https://github.com/nakkstar123/grover>. Our implementation uses the publicly available implementation of Duty-Free Bits [1] as a dependency. The complex-multiplication optimization of Appendix A is not implemented. We report the size of the garbled program and validate the implementation end to end against Groth16 proofs generated by `arkworks` [4].

Parameters. We set

$$n = \lceil \log_2 r \rceil = 254, \quad \kappa = \lambda = 128, \quad \tau = 2\lambda = 256.$$

With these parameters, the field encoding dimensions are as follows:

$$S_A^{\mathbb{F}} = 12n + 5 = 3053, \quad S_B^{\mathbb{F}} = S_C^{\mathbb{F}} = 12\kappa + 5 = 1541.$$

For A and C , the prime-field dimension is simply $S_A^{\mathbb{F}}$ and $S_C^{\mathbb{F}}$. For B , the additive-sharing construction of Lemma 12 contributes four prime-field coordinates for every coordinate of the original $\mathbb{F}_{p^2}$ encoding. Hence

$$N_{\text{base}} = S_A^{\mathbb{F}} + 4S_B^{\mathbb{F}} + S_C^{\mathbb{F}} = 10,758.$$

We target 40 bits of statistical privacy, as is standard in our setting. Duty Free Bits [17] sets parameters such that the the protocol has $< 2^{-40}$ privacy loss. In our setting, we use the first 82 primes while constructing the CRT modulus M . The largest is 421, and their product satisfies $\log_2 M \approx 569.6$. Setting $\rho = 54$ gives

$$N_{\text{base}} 2^{-\rho} = 10,758 \cdot 2^{-54} < 2^{-40}.$$

which is an upper bound on the aggregate statistical privacy loss when using Theorem 11, as proved in [17].

Communication. The garbled program’s total size is 2501 KiB (2.443 MiB): we split this into the group program $\tilde{f}_{\text{G}}$ and its projectivization via Π_2 ’s eight Duty-Free Bits instances [17] applied to the ITPG encodings of [11, 13] which dominates the cost.

Component		Size
Group program $\tilde{f}_{\text{G}}$		12 KiB
Projectivization	Fixed	885 KiB
	Variable	1604 KiB
Total		2501 KiB (2.443 MiB)

Projectivizing splits into a fixed cost, the one-time conversion of each instance’s 254-bit input into CRT residues, independent of how many affine map dimension of that input, and a variable cost, proportional to $N_{\text{base}} = 10,758$, the aggregate output dimension. Duty-Free Bits [17] suggests an optimization, not implemented here, for eliminating output decoding masks that roughly halves the variable cost, bringing the total cost down to an estimated ≈ 1.6 MiB.

End-to-end validation. We test the complete implementation using a Groth16 proof generated by `arkworks` over BN254. The test suite also verifies that off-curve coordinates and a curve point B outside $\mathbb{G}_2$ lead to correct secret disclosure. Finally, it checks that non-canonical 256-bit coordinate strings are reduced modulo p and agree with their canonical representatives.

Comparison with Boolean garbling. Mosaic [18] reports that privacy-free half-gate garbling of the state-of-the-art three-billion-AND-gate Groth16 verifier produces a 48 GB garbled program. At 2.443 MiB, our garbled program is approximately 20,000 $\times$ smaller than the Boolean baseline, which is the only public implementation of Groth16 garbling, and therefore prior state of the art. In Appendix A, we estimate the size of the resulting garbled program if we incorporate further optimizations.

References

- [1] Alpen Labs. Duty-free bits: Reference implementation and architecture, 2026. Revision e4eecc41d9fe. <https://github.com/alpenlabs/duty-free-bits>.
- [2] Alpen Labs. Groth16 boolean verifier circuit. Software repository, 2026. <https://github.com/alpenlabs/g16>.
- [3] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press. Preprint: <https://eprint.iacr.org/2012/255>.
- [4] arkworks contributors. arkworks zkSNARK ecosystem. <https://arkworks.rs>, 2022.
- [5] Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 3–34, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland. Preprint: <https://eprint.iacr.org/2023/501>.
- [6] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 565–577, Vienna, Austria, October 24–28, 2016. ACM Press. Preprint: <https://eprint.iacr.org/2016/969>.
- [7] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331, Kingston, Ontario, Canada, August 11–12, 2006. Springer Berlin Heidelberg, Germany. Preprint: <https://eprint.iacr.org/2005/133>.
- [8] BitVM contributors. Garbled SNARK verifier (BitVM). <https://github.com/BitVM/garbled-snark-verifier>. GitHub repository. Accessed September 17, 2026.
- [9] Vitalik Buterin and Christian Reitwiessner. EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. Ethereum Improvement Proposals, no. 197, 2017. <https://eips.ethereum.org/EIPS/eip-197>.

- [10] Liam Eagen. Glock: Garbled locks for Bitcoin. Cryptology ePrint Archive, Paper 2025/1485, 2025. <https://eprint.iacr.org/2025/1485>.
- [11] Liam Eagen and Ying Tong Lai. Argo MAC: Garbling with elliptic curve MACs. Cryptology ePrint Archive, Paper 2026/049, 2026. <https://eprint.iacr.org/2026/049>.
- [12] Sanjam Garg, Mohammad Hajiabadi, Dimitris Kolonelos, Abhiram Kothapalli, and Guru-Vamsi Policharla. A framework for witness encryption from linearly verifiable SNARKs and applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 504–539, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland. Preprint: <https://eprint.iacr.org/2025/1364>.
- [13] Sanjam Garg, Dimitris Kolonelos, Mikhail Sergeevitch, Srivatsan Sridhar, and David Tse. BABE: Verifying proofs on Bitcoin made 1000x cheaper. Cryptology ePrint Archive, Paper 2026/065, 2026. <https://eprint.iacr.org/2026/065>.
- [14] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany. Preprint: <https://eprint.iacr.org/2016/260>.
- [15] David Heath. Efficient arithmetic in garbled circuits. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 3–31, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland. Preprint: <https://eprint.iacr.org/2024/139>.
- [16] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014: 41st International Colloquium on Automata, Languages and Programming, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 650–662, Copenhagen, Denmark, July 8–11, 2014. Springer Berlin Heidelberg, Germany. Preprint: <https://eprint.iacr.org/2014/995>.
- [17] Nakul Khambhati, Anwesh Bhattacharya, and David Heath. Duty-free bits: Projectivizing garbling schemes. Cryptology ePrint Archive, Paper 2026/476, 2026.
- [18] Nakul Khambhati, Mukesh Tiwari, Azz, Sapin Bajracharya, Manish Bista, Liam Eagen, Christian Lewe, and Aaron Feickert. Mosaic: Practical malicious security for garbled circuits on Bitcoin. Cryptology ePrint Archive, Paper 2026/812, 2026. <https://eprint.iacr.org/2026/812>.
- [19] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer Berlin Heidelberg, Germany. https://www.cs.utoronto.ca/~vlad/papers/XOR_ICALP08.pdf.
- [20] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part IV*, volume 15367 of *Lecture Notes in Computer Science*, pages 71–97, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland. Preprint: <https://eprint.iacr.org/2024/820>.

- [21] Joost Renes, Craig Costello, and Lejla Batina. Complete addition formulas for prime order elliptic curves. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 403–428, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany. Preprint: <https://eprint.iacr.org/2015/1060>.
- [22] Jeremy Rubin. Delbrag. Draft, April 4, 2025. <https://rubin.io/public/pdfs/delbrag.pdf>.
- [23] Mikhail Sergeevitch, Konrad Staniec, David Tse, Nikhil Vanjani, and Robin Linus Woll. Non-interactive translation of Winternitz signatures to Lamport signatures via secret sharing. Cryptology ePrint Archive, Paper 2026/1684, 2026.
- [24] Mukesh Tiwari and Aaron Feickert. Antichain Winternitz: Guaranteed garbled-circuit label revelation on Bitcoin with permissionless recovery. Cryptology ePrint Archive, Paper 2026/1568, 2026.
- [25] Robin Linus Woll, Ioannis Alexopoulos, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and David Tse. BitVM3: Efficient Bitcoin bridges via garbled circuits. Cryptology ePrint Archive, Paper 2026/933, 2026. <https://eprint.iacr.org/2026/933>.
- [26] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Optimization: Complex Multiplication on BN254

The BN254 elliptic curve used in our construction admits multiple efficient complex-multiplication endomorphisms. These can be used, as partly described by Argo MAC [11], to lower the overall cost of executing our Groth16 garbling protocols.

Specifically, we can apply these endomorphisms to shorten the random coefficient vector used to recover the secret used in disclosure, which in turn reduces the garbling material required for later evaluation and disclosure. To do this, we first describe the changes required to $\Pi_{\mathbb{G}}$ in garbling, encoding, and evaluation. Additionally, we must move from the incomplete addition formulas presented in Section 3 to complete addition formulas, which we describe here.

We note an additional related optimization is possible. We can replace the binary scalar decomposition used in the rekeying gadget with a decomposition against the endomorphism set. Such a decomposition is always possible in the BN254 setting, and it is possible to show numerically that such a decomposition can be done more compactly than in the binary context. Despite this, the analysis involved in showing the bound is nontrivial. As the resulting overall reduction in protocol cost is marginal, we defer this optimization.

A.1 Endomorphisms and $\Pi_{\mathbb{G}}$

Choose primitive cube roots $\zeta \in \mathbb{F}_p$ and $\omega \in \mathbb{F}_r$ such that

$$\psi_1(u, v) := (\zeta u, v), \quad \psi_2(u, v) := (\zeta^2 u, v)$$

act as multiplication by ω on $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, acting coordinatewise against affine point representations. These are known to exist for our specific instantiation over BN254.

From these roots, we set

$$\Sigma_{\text{CM}} = \{0, \pm 1, \pm \omega, \pm \omega^2\}.$$

For each $c = \epsilon \omega^j$ such that $\epsilon \in \{\pm 1\}$ and $j \in \{0, 1, 2\}$, define

$$\phi_c^{(1)}(u, v) := (\zeta^j u, \epsilon v) = \epsilon \psi_1^j$$

on $E(\mathbb{F}_p)$, and

$$\phi_c^{(2)}(u, v) := (\zeta^{2j} u, \epsilon v) = \epsilon \psi_2^j$$

on $E'(\mathbb{F}_{p^2})$. Let $\phi_0^{(1)}$ and $\phi_0^{(2)}$ be each corresponding zero map. These are defined precisely such that on corresponding source group $\mathbb{G}_i$, the function $\phi_c^{(i)}$ is simply multiplication by c .

In the construction $\Pi_{\mathbb{G}}$ described in Section 5, we now sample each c_t uniformly from Σ_{CM} for $t \in [\kappa]$ during garbling. We then replace multiplication of a point by c_t with evaluation of the point against the corresponding $\phi_c^{(i)}$ (and do so when encoding as well).

Evaluation also changes to account for the endomorphisms. In the case where $rB = \mathcal{O}$, the evaluator now checks candidate values $d_t \in \Sigma_{\text{CM}}$ to match the hash value h_t . In the case where $rB \neq \mathcal{O}$, the evaluator similarly now must check candidate values $d_t \in \Sigma_{\text{CM}}$ such that

$$\phi_{d_t}^{(2)}(rB) = r\llbracket B \rrbracket_t.$$

However, this is only uniquely defined if these values do not collide across the endomorphism set. This fact is established in [11, Section 3.2.3], so d_t is always uniquely determined in this case.

The results established in Section 5 are modified in a natural way to account for these changes in $\Pi_{\mathbb{G}}$.

A.2 Complete Addition

The proof of Theorem 3 describes incomplete Jacobian addition laws for coordinates of points of the form $cV + K$. In order to move to the use of the expanded endomorphism set Σ_{CM} , we also move to complete addition formulas to account for all input cases comprehensively.

To do so, we use the complete projective addition formulas of Renes, Costello, and Batina [21] applied to the BN254 source groups. While they incur higher cost in terms of encodings, we will show that this cost is offset at the protocol level.

The formulas presented use homogeneous coordinates where $(X : Y : Z)$ represents the scaled affine point $(X/Z, Y/Z)$. We therefore consider an affine input point $V = (u, v)$ represented as $(u : v : 1)$ projectively, and suppose that for some $0 \neq c \in \Sigma_{cm}$, we express $\phi_c(V) = (\alpha u, \sigma v)$ for α, σ depending on c as defined above. Further, suppose the mask is $K = (a : b : z)$, and choose β such that each curve (E and the twist E') is defined via the equation $v^2 = u^3 + \beta$.

Using this notation, we can obtain complete addition formulas for $\phi_c(V) + K$ from the cited work by simple substitution:

$$\begin{aligned} X &= \alpha\sigma(b^2 - 3\beta z^2)uv + a\sigma^2bv^2 - 6\beta\alpha bz u - 6\beta a\sigma z v - 3\beta ab, \\ Y &= \sigma^2b^2v^2 + 9\beta\alpha^2az u^2 + 9\beta\alpha a^2u - 9\beta^2z^2, \\ Z &= \sigma^2bzv^2 + \sigma(b^2 + 3\beta z^2)v + 3\beta bz + 3\alpha^2ab u^2 + 3\alpha\sigma a^2uv. \end{aligned} \tag{2}$$

The sum-of-monomials ITPG now computes $(\mu X, \mu Y, \mu Z)$ for uniformly-sampled $\mu \in \mathbb{F}^\times$ over the relevant coordinate field.

As before, we simply sum the dimensions of u and v appearing in the output coordinates to obtain the dimensions of the corresponding encoding vectors. This gives dimension 8 for u and 10 for v .

A.3 Estimated cost

Keeping the hash and statistical parameters of the main instantiation, we estimate the cost of the optimized construction using $\kappa = \lceil 128/\log_2 7 \rceil = 46$. Given the enlarged endomorphism set, this encoding vector dimension suffices to obtain $\lambda = 128$ bits of entropy. The encoding of A is unchanged, while those of B and C use the complete addition formulas above. Including the curve-equation checks gives

$$S_A^{\mathbb{F}} = 12 \cdot 254 + 5 = 3053, \quad S_B^{\mathbb{F}} = S_C^{\mathbb{F}} = 18 \cdot 46 + 5 = 833,$$

and hence an aggregate affine encoding dimension of

$$N_{cm} = S_A^{\mathbb{F}} + 4S_B^{\mathbb{F}} + S_C^{\mathbb{F}} = 7218,$$

down from 10,758.

The eight Duty-Free Bits instances retain the same fixed cost, while their variable cost scales proportionally to the affine encoding dimension. This results in the following estimated garbled program size.

Component		Size
Group program $\tilde{f}_{\mathbb{G}}$		4 KiB
Projectivization	Fixed	885 KiB
	Variable	1076 KiB
Total		1965 KiB (1.92 MiB)

This is approximately 21% below the binary implementation’s 2.443 MiB. Although the coordinate encoding dimension for B and C increases from 12 to 18 per point, reducing κ from 128 to 46 lowers their total cost.