Duty-Free Bits: Projectivizing Garbling Schemes*

Nakul Khambhati
nakul@cs.ucla.edu
UCLA, Alpen Labs

Anwesh Bhattacharya
anwesh2@illinois.edu
UIUC

David Heath
daheath@illinois.edu
UIUC

Abstract

Garbling schemes are powerful primitives that enable secure computation between a mutually untrusting *garbler* and *evaluator*. A *projective* garbling scheme is one that encodes the evaluator's input in a simple bit-by-bit manner. Projective schemes, such as the seminal scheme of Yao, are versatile, as they are naturally compatible with other simple tools, such as 1-out-of-2 oblivious transfer (OT).

There exist garbling schemes that naturally operate over large finite fields, some of which require only efficient information-theoretic (IT) techniques. However, here the evaluator's input is encoded via an affine function over a large field, so these schemes are not naturally projective, reducing their versatility.

We provide a transformation that efficiently *projectivizes* such schemes. Consider an arithmetic garbling scheme where the evaluator's input consists of elements from a large prime field. Our symmetric-key-based techniques give a mechanism to translate from Yao-style labels to IT-style labels at cost *proportional to the input and output labels*: crossing the border is duty-free!

We apply our technique to two problems. First, recent works show that projective garbling schemes solve a problem central to *trust-minimized bridges* for the Bitcoin blockchain, where BABE (Garg et al., 2026) and Argo MAC (Eagen and Lai, 2026) each implicitly construct an efficient IT garbling scheme, then use naive bit-decomposition to achieve projectivity. We construct and implement drop-in replacements for both; we improve BABE's encoding size by 51×, and Argo MAC's by 23×. Second, our technique implies a non-interactive reduction from vector oblivious linear evaluations (VOLEs) over $\mathbb{F}_p$ to 1-out-of-2 OTs. This is the state-of-the-art Minicrypt (plus base OTs) protocol for large field VOLE secure against a malicious receiver, and a length-S VOLE costs only $O((\lambda + S) \lg p)$ bits.

Keywords: Garbled Circuits, Projectivity, Bitcoin Bridges, Vector OLE

*An extended abstract of this work appears in the proceedings of ACM CCS 2026, https://doi.org/10.1145/3830454.3846724. This is the full version.

Contents

1 Introduction **3**
- 1.1 Our Contribution . . . 4
- 1.2 Applications . . . 4
- 1.3 Intuition for Our Approach . . . 5

2 Notation **6**

3 Preliminaries **7**
- 3.1 Garbling Schemes . . . 7
- 3.2 Cryptographic Assumption . . . 8
- 3.3 Switch Systems . . . 8
- 3.4 Statistical Smudging . . . 9

4 Overview **9**
- 4.1 Projectivizing Affine Garbling Schemes . . . 10
- 4.2 Projectivizing Affine Maps . . . 10
- 4.3 Affine Evaluation via CRT Residues . . . 10

5 Projectivizing Garbling Schemes **11**
- 5.1 Projective Partial Garbling Scheme for Affine Maps over Prime Fields . . . 12
- 5.2 From Affine Encodings to Projectivity . . . 12

6 Switch System for Affine Maps **13**
- 6.1 Switch Systems with IT Labels . . . 13
- 6.2 Helper Switch Systems . . . 16
- 6.3 Affine Maps over CRT Rings . . . 17
  - 6.3.1 Join-width . . . 18

7 Applications **18**
- 7.1 Argo MAC Encoding . . . 19
- 7.2 Embryo: Garbled Scalar Multiplication . . . 19
- 7.3 $\mathbb{F}_p$-VOLE Secure Against a Malicious Receiver . . . 20

8 Implementation **20**

A Garbled Switch System Construction **25**

B Hash-Optimized Word-to-Ring Conversions **28**

C Elliptic Curve Preliminaries **29**
- C.1 Notation . . . 29
- C.2 Elliptic Curve Addition . . . 30

D Application Constructions and Proofs **31**
- D.1 Embryo . . . 31
- D.2 $\mathbb{F}_p$-VOLE . . . 34

E Related Work on Vector OLE **35**

F Concrete Cost Analysis **35**

1 Introduction

Garbling schemes are powerful cryptographic primitives that enable a form of privacy-preserving outsourced computation. Consider a function f that takes inputs x and y. Garbling schemes allow two parties—an *evaluator* E with input x and a *garbler* G with input y—to factor computation of f into several steps. In this work, we consider the following flow. G uses his input y to sample a *garbling* $\tilde{f}$, as well as some encoding data e. The parties use e to *encode* E's input x, yielding $\tilde{x}$. Finally, G sends $\tilde{f}$ to E, who uses x, $\tilde{f}$, and $\tilde{x}$ to locally *evaluate*, yielding $f(x, y)$. This flow—where G knows y at garbling-time and E holds x in cleartext—is a *partial* garbling scheme; see Section 3.1. The main security property is *privacy*: the pair $(\tilde{f}, \tilde{x})$ should hide G's input y from E.

Garbling schemes have numerous cryptographic applications, including constant-round secure two-party and multiparty computation (2PC and MPC), and more generally serve as powerful primitives for building higher-level cryptographic objects. Because of their versatility, garbling schemes enjoy intense research focus, and the literature is rich with techniques for building them, trading off simplicity, efficiency, underlying assumptions, and expressivity.

The most well-known garbling technique is that of the seminal work of Yao, which was improved by a long line of work [Yao86, BMR90, NPS99, KS08, PSSW09, KMR14, ZRE15, GLNP18, RR21]. This line of work uses symmetric-key-style assumptions to garble any function expressed as a Boolean circuit C.

One crucial advantage of Yao's scheme is the *simplicity* of how E's input x is encoded: for each bit x_i, G chooses two *labels*, one for $x_i = 0$ and one for $x_i = 1$, and encoding E's input simply involves choosing the label corresponding to x_i. This property—which the literature sometimes refers to as **projectivity** [BHR12b]—is invaluable when using a garbling scheme to build higher-level objects. For instance, a projective scheme can be straightforwardly transformed into a constant-round 2PC protocol by using 1-out-of-2 oblivious transfer (OT) to allow E to choose her input labels. As a second example, projectivity allows garbling schemes to be used in contexts where the per-wire labels are separately *committed*, since only a feasible number of commitments are needed. This has recently been leveraged to achieve so-called *trust-minimized bridges* for the Bitcoin blockchain; see [LAA+25, Che25, Eag25] and Section 7 for more discussion.

The crucial *disadvantage* of Yao's scheme is the size of its encodings. It is generally agreed that the size of encodings $(\tilde{f}, \tilde{x})$ is the most important performance criterion for garbling schemes, as this size imposes communication complexity for higher-level protocols. To garble a Boolean circuit C, the line of work on Yao's scheme produces encodings of size $O(|C|\lambda)$ bits, where λ is a computational security parameter, namely the bit-length of encryption keys. This cost is problematic in two ways. The factor-λ blow-up is itself undesirable, as each AND gate imposes an encoding of size ≈ 24 bytes or more [RR21]. Moreover, many natural functions f compile to large Boolean circuits C in the first place. Arithmetic operations, for instance, incur blow-up once compiled to low-level Boolean gates, even using the asymptotically best algorithms [HvdH21].

Amazingly, there exist simple, information-theoretically-secure garbling schemes (IT-GSs) that avoid both problems above, for certain functions f [IK00, IK02, CFIK03, IW14, App17]. For example, consider the case where f takes as input elements from some prime field $\mathbb{F}_p$ and computes a constant-degree *arithmetic formula* (i.e., a circuit with no fan-out) over the input. In this case, there exists a simple IT-GS whose total encoding size is only $O(|f|)$ $\mathbb{F}_p$ elements, more than a factor λ improvement as compared to Yao's scheme!

Unfortunately, these IT-GSs have a crucial shortcoming, which is our focus in this work: **IT-GSs are not projective.** Indeed, for IT-GS schemes that operate over finite fields $\mathbb{F}_p$, E's input x is now itself a vector of $\mathbb{F}_p$ elements, and encoding each element x_i now involves computing $\mathbf{a}x_i + \mathbf{b}$, where $\mathbf{a}, \mathbf{b}$ are vectors of $\mathbb{F}_p$ elements chosen by G. This is problematic for applying such techniques. For instance, we can no longer use a single 1-out-of-2 OT to encode each element x_i. One could consider encoding E's input by replacing OT by *vector oblivious linear evaluation* (vector OLE, VOLE), but VOLE protocols are less primitive and costlier. In addition, one might hope to use a garbling scheme to *build* a VOLE protocol, in which case using VOLE to transmit input labels would be circular. Indeed, we use our techniques to reduce VOLE to OT; see Section 7.

It is thus natural to consider augmenting such IT-GSs with projectivity, enjoying the benefits of IT-GS's smaller encodings *and* the flexibility of Yao's scheme. Projectivizing an IT-GS for $f(x, y)$ over $\mathbb{F}_p$ involves building a new garbling scheme for the function $f'(x', y) = f(\mathsf{bin\text{-}to\text{-}}\mathbb{F}_\mathsf{p}(x'), y)$, which takes E's input x' as a bitstring rather than as $\mathbb{F}_p$ elements ($\mathsf{bin\text{-}to\text{-}}\mathbb{F}_\mathsf{p}$ reinterprets the bits as field elements in the natural manner). The function f', too, can be securely computed by an IT-GS: take as input E's bits x', each encoded as a

field element from $\{0,1\} \subseteq \mathbb{F}_p$, compute linear operations over these to recover E's desired input x, then use the original IT-GS to compute f. To our knowledge and modulo constant factors, this transformation is the best prior approach for upgrading IT-GSs with projectivity. It is also expensive: it multiplies the size of IT-GS encodings by factor $\lg p$, which is clearly undesirable, especially when working with a large field $\mathbb{F}_p$. Encodings of S field elements become $S \lg p$ field elements.

As one illustrative example of this cost, [GKS+26]'s carefully-tuned garbling of scalar multiplication over BN254 admits an IT-GS encoding of size less than 95 KiB, yet their scheme produces encodings of size more than 22 MiB, a $\approx 240\times$ increase. This blow-up is due primarily to their application requiring *projectivity*; see Section 1.2.

1.1 Our Contribution

We provide an asymptotically- and concretely-improved technique for upgrading garbling schemes with projectivity. Consider a garbling scheme that computes a function $f(x, y)$, where E's input x is encoded as elements over a large field $\mathbb{F}_p$. Suppose the scheme's encodings $(\tilde{f}, \tilde{x})$ are such that $\tilde{x}$ consists of S field elements, and can be computed from x via component-wise affine functions. Given such a scheme, we construct a new *projective* garbling scheme computing the same function with encodings $\tilde{f}', \tilde{x}'$ such that

1. $|\tilde{f}'|$ is at most $O((|x|\lambda + S) \lg p)$ bits larger than $|\tilde{f}|$.
2. $\tilde{x}'$ consists of $|x| \lg p$ Yao-style labels, each of length λ and encoded via a simple selection.

Thus, for example, if $|x|$ is a constant and if $\lambda = O(S)$, our encoding size increases by only $O(S \lg p)$ bits. Our scheme's encodings are also concretely small: at BN254 parameters, the increase is ≈ 110 KiB per input element plus 593 bits per encoding element (Section 8). See Section 1.2 for a brief discussion.

Our scheme is proven secure assuming a circular-correlation robust hash function (CCR hash, CCRH). CCR hashes are leveraged extensively in the garbling scheme literature, as they enable security of the celebrated Free XOR optimization [KS08, CKKZ12].

At a technical level, our garbling scheme solves the following problem: Given vectors $\mathbf{a}_i, \mathbf{b}_i \in \mathbb{F}_p^S$ from the garbler and scalar $x_i \in \mathbb{F}_p$ from the evaluator, where x_i is written in binary, compute $\mathbf{a}_i x_i + \mathbf{b}_i$. By computing such an affine map for each element in E's input, we deliver *all* IT-GS labels to E, which she subsequently uses to evaluate $f(x, y)$. Our scheme computes $\mathbf{a}_i x_i + \mathbf{b}_i$ at communication cost only $O((\lambda + S) \lg p)$ bits. Note that this additive term is *proportional to the size of* the Yao-style input labels and IT-GS output labels! Therefore, we describe this conversion as *duty-free*.

A familiar reader may notice that computing $\mathbf{a}_i x_i + \mathbf{b}_i$ is the ideal functionality of vector OLE (VOLE) over $\mathbb{F}_p$, where G is the VOLE sender and E is the VOLE receiver. Indeed, our scheme immediately implies an $\mathbb{F}_p$ VOLE protocol secure against a malicious receiver.

We implement our scheme—garbling and evaluation—alongside a bit-decomposition baseline, and measure communication, hash counts, and end-to-end runtime (Section 8).

1.2 Applications

Trust-Minimized Bridges for Bitcoin. Recent works [EL26, GKS+26] have demonstrated the relevance of IT-GSs to *trust-minimized bridges* for the Bitcoin blockchain. The central challenge is that Bitcoin's scripting language is too inexpressive to verify arbitrary computation—such as a Groth16 SNARK proof [Gro16]—on-chain. The BitVM line of work [LAA+25, Che25, Eag25] addresses this via an optimistic protocol: G commits to a garbled SNARK verifier off-chain, and if G later submits an invalid proof, E uses the pre-shared garbled circuit to generate a fraud proof, shifting the verification burden off-chain.

A crucial constraint of this setting is that E's garbled input must be authenticated on-chain via Lamport signatures. In a Lamport signature on a field element $x \in \mathbb{F}_p$, E reveals, for each bit x_j of x, one of two λ-bit preimages $L_{j,0}$ or $L_{j,1}$. This coincides exactly with a projective input encoding: the revealed preimages serve simultaneously as the Lamport signature on-chain and as the garbled input labels off-chain. Projectivity is therefore not merely convenient, it is mandated by the on-chain authentication mechanism.

Both [EL26] and [GKS+26] construct efficient IT-GSs for their respective computations, then projectivize via bit-decomposition, incurring an $\lg p$-fold blowup in encoding size. BABE [GKS+26] garbles a scalar

multiplication $(\pi, r) \mapsto r \cdot \pi$ over BN254, where G holds a private scalar r and E holds a curve point π. By combining with a simple witness encryption scheme for linear pairing relations, [GHK+25] show how to verify Groth16 proofs on Bitcoin. Their IT-GS outputs $S = O(\lg p)$ encodings over $\mathbb{F}_p$; after bit-decomposition, the garbled encoding is 22.16 MiB. [EL26] (Argo MAC) garbles the computation of $\kappa = O(\lambda)$ homomorphic MAC components on a curve point, each a low-degree polynomial evaluation on the point's affine coordinates; their IT-GS has $S = O(\kappa)$ encoding elements, and after bit-decomposition produces an encoding of 6.8 MiB.[1]

Our technique serves as a drop-in replacement for the projectivization step in both works. For scalar multiplication, our construction—which we call Embryo—reduces the garbled encoding to $\approx$ 442 KiB, a 51× improvement. For Argo MACs, we reduce the encoding to $\approx$ 298 KiB, a 23× improvement. The reduction in size costs computation mainly on the evaluator's side: our garbler performs about twice the baseline's work, while our evaluator performs about 3.6× the hash calls per encoding element (Section 8). At 100 Mbps, end-to-end time nevertheless improves by 22×. Embryo incorporates an input validity check at a cost of only $O(\lg p)$ bits; BABE handles this check via a Yao-style garbled circuit, incurring unnecessary overhead. The constructions are presented in Section 7, with costs in Section 8 and an analytic cost breakdown in Appendix F.

Vector OLE Over Large Fields. Oblivious linear evaluation (OLE) and vector OLE (VOLE) can be understood as generalizations of OT. In OLE, a sender holds a, b and a receiver holds x; as output, the receiver obtains $ax + b$. Viewed another way, the parties ultimately hold additive secret shares of ax. VOLE generalizes this to the setting where the sender holds two *vectors* $\mathbf{a}, \mathbf{b}$, and the receiver outputs $\mathbf{a}x + \mathbf{b}$. VOLE and OLE are well-studied primitives with numerous applications, including arithmetic MPC protocols [ADI+17], zero-knowledge proofs [BDSW23], and oblivious pseudorandom function/private set intersection protocols [RS21].

Our technique immediately implies a Minicrypt reduction from large field VOLE to OT where each OLE incurs amortized cost $O(\lg p)$ bits. Namely, we *non-interactively* reduce a length-S VOLE to $\lg p$ OTs in zero rounds, at cost $O((\lambda + S) \lg p)$ bits. Indeed, recall that computing $\mathbf{a}x + \mathbf{b}$ for x expressed in bits is essentially our main technical challenge. The reduction is non-interactive: after the OT calls, the receiver sends no further messages. It is therefore naturally secure against a malicious receiver (Section 7.3). To our knowledge, ours is the state-of-the-art Minicrypt reduction from large field VOLE to OT with security against a malicious receiver.

Section 7.3 and Appendix E discuss related work on OLE/VOLE and its relation to our approach.

1.3 Intuition for Our Approach

The starting point for our work is the one-hot garbling technique of [HK21]. In short, there exists inherent friendliness between symmetric-key-based garbling and *one-hot vectors*, whose entries are all zeros except a single *active* entry holding a one. In a garbled circuit (and assuming E knows in cleartext the location of the active entry), such vectors can be efficiently constructed and manipulated [HK21].

Indeed, prior work observes that given a Free-XOR-style garbling of a one-hot vector $\mathbf{h} \in \{0,1\}^d$ and the Free-XOR-style garbling of a scalar bit s, it is possible for G and E to jointly construct a garbling of the vector-scalar product $s\mathbf{h}$ at cost only *a single ciphertext*. That is, the size of the garbling of this building-block operation is *independent* of the length of $\mathbf{h}$.

A main insight of our work is that this efficient vector-scalar operation can be generalized to scale a one-hot vector by a field element chosen by the garbler G.

To see this generalization, let us for this section suppose that H is a random oracle that outputs elements in $\mathbb{F}_p$. Let $\mathbf{h} \in \{0,1\}^d$ be a one-hot vector with active position x, and let $a \in \mathbb{F}_p$ be a secret field element chosen by G. Suppose each entry i of the one-hot vector has two possible garbled labels: L_i^0, corresponding to $h_i = 0$, and L_i^1, corresponding to $h_i = 1$. Thus, E holds $L_{i \neq x}^0$ and L_x^1; G knows all labels L_i^b.

To encode the vector-scalar product, G can send to E the following single ciphertext:

$$\mathrm{ct} = \left(\sum_i H(L_i^0, \mathit{nonce}) \right) + a$$

[1] The size of this encoding depends on the number κ of MAC components generated and the underlying elliptic curve group. We compare with $\kappa = \lceil \lg p / \log_2 7 \rceil$ (see [EL26] for more details) and the BN254 elliptic curve group.

Notice that this ciphertext hides a from E. Indeed, E does not know L_x^0, so $H(L_x^0, nonce)$ acts as a one-time pad. Notice also that E, who holds each zero label $L_{i\neq x}^0$, can locally compute $H(L_{i\neq x}^0, nonce)$ and subtract these from ct, obtaining $H(L_x^0, nonce) + a$. Thus, we have computed an IT-GS-style encoding of the vector-scalar product $s\mathbf{h} \in \mathbb{F}_p^d$: the evaluator holds each zero label $H(L_{i\neq x}^0, nonce) \in \mathbb{F}_p$, and at the active position x, she holds a label corresponding to a, namely $H(L_x^0, nonce) + a \in \mathbb{F}_p$. To compute this, the garbler sent only a single field element.

Recall, our main technical goal is to compute $\mathbf{a}x + \mathbf{b}$ for secret vectors of field elements $\mathbf{a}$ and $\mathbf{b}$ chosen by the garbler and for scalar $x \in \mathbb{F}_p$ garbled using Yao-style labels and known in clear to the evaluator; for simplicity, focus on scalars $a, b \in \mathbb{F}_p$. If we worry only about communication and temporarily set aside runtime, here is an approach that would work. First, convert the Yao-style encoding of x to a garbled one-hot encoding, i.e. a length-p vector $\mathbf{h}$ whose x-th entry is a one, at cost $O(\lg p\lambda)$ bits. Second, scale $\mathbf{h}$ by a as described above, at the cost of one field element. Third, use IT-GS labels' support for linear operations to locally compute an encoding of $\sum_i i \cdot (a\mathbf{h}) = ax$. The resulting label has form $ax + r$ for $r = \sum_i i \cdot H(L_i^0)$, so the garbler sends $b - r$, and the evaluator adds it to obtain $ax + b$. To extend to the vector $\mathbf{a}x + \mathbf{b} \in \mathbb{F}_p^S$, re-use the one-hot conversion and repeat the remaining steps per index, for total cost $O(\lg p\lambda + S \lg p)$ bits.

The above captures an insight of our approach: leverage the compatibility of garbled one-hot encodings and IT-GS-style labels to efficiently compute the necessary affine transformation. However, the above template does not work, because the field size p can be exponential: the communication is very low, but the *computation* is infeasible. To fix this, we leverage and extend additional tricks from the literature on symmetric-key-based arithmetic garbled circuits [Hea24]. Rather than computing $\mathbf{a}x + \mathbf{b}$ in the field $\mathbb{F}_p$, we instead compute it (plus a statistically-smudging mask for security) in an integer ring $\mathbb{Z}_M$ with M "big enough" to prevent modular wrap-around. By the Chinese Remainder Theorem (CRT), this is equivalent to computing $\mathbf{a}x + \mathbf{b} \bmod p_i$ for various *small primes* p_i. Because each prime is small, the procedure above becomes computationally efficient. Section 4 provides a more detailed high-level explanation of our approach, and Sections 5 and 6 provide the formal treatment.

2 Notation

We write λ for a computational security parameter (e.g. the length of hash outputs or encryption keys) and ρ for a statistical security parameter. $\mathbb{F}_p$ denotes a prime field. We often use $n := \lg p$ to denote the bit-length of field elements, and in our Bitcoin application, $n = O(\lambda)$. We use lg to denote the *ceiled* base-two logarithm: $\lg(\cdot) = \lceil \log_2(\cdot) \rceil$. G denotes the garbler (referred to as "he"); E denotes the evaluator (referred to as "she"). We use S for the total number of field elements in the underlying garbling scheme's encoding of E's input.

CRT parameters. Let $p_1 < p_2 < \cdots$ denote the sequence of primes ($p_1 = 2$). Fix n, and define $M := \prod_{t=1}^{T} p_t$ to be the first primorial satisfying $M \geq 2^{2n+\rho}$, and set $m := \lg M$ (so $m \geq 2n + \rho$). By the prime number theorem, $p_T = \Theta(m) = \Theta(n)$ and $T = \Theta(m/\log m) = \Theta(n/\log n)$. We write $\mathbb{Z}_M \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_T}$ for the corresponding product ring via the Chinese Remainder Theorem.

Representations. For $x \in \{0, \ldots, 2^n - 1\}$, we write $\mathsf{bin}(x) = (x_0, \ldots, x_{n-1}) \in \{0,1\}^n$ for its binary decomposition, so that $x = \sum_{j=0}^{n-1} 2^j x_j$. Conversely, for a bitstring $\mathbf{x} \in \{0,1\}^n$, we write:

$$\mathsf{bin\text{-}to\text{-}}\mathbb{F}_\mathsf{p}(\mathbf{x}) = \sum_{j=0}^{n-1} 2^j x_j \bmod p \in \mathbb{F}_p.$$

Also, we denote by $\mathbf{x} = (x_1, \ldots, x_{|\mathbf{x}|})$ a vector of ring/field elements with $|\mathbf{x}|$ being the length of the vector.

3 Preliminaries

3.1 Garbling Schemes

We use the following definition of a *partial* garbling scheme, derived from definitions in [BHR12b, IW14]. The notion of a *partial* scheme—as opposed to the "full security" notion—corresponds to the fact that the evaluator E knows some of the input to the garbled program in cleartext, which maps well to our setting: our main goal is to construct a garbling scheme that generates an encoding of E's input, which E knows.

Definition 1 (Partial Garbling Scheme)**.** *A **partial garbling scheme** for a function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is a triple of algorithms* $(\mathsf{Garb}, \mathsf{Enc}, \mathsf{Eval})$*, where:*

- Garb *takes a security parameter 1^λ, and a garbler input $y \in \mathcal{Y}$. It outputs an encoding $\tilde{f}$, and an input encoding string e.*
- Enc *takes an input encoding string e and an evaluator input $x \in \mathcal{X}$. It outputs an encoding $\tilde{x}$.*
- Eval *takes a program encoding $\tilde{f}$ and an input encoding $\tilde{x}$. It outputs a value $z \in \mathcal{Z}$.*

*The scheme is **correct** if for all inputs $x \in \mathcal{X}, y \in \mathcal{Y}$ the following holds with overwhelming probability in λ:*

$$\Pr\left[f(x,y) = z \;\middle|\; \begin{array}{l} (\tilde{f}, e) \leftarrow \mathsf{Garb}(1^\lambda, y) \\ \tilde{x} \leftarrow \mathsf{Enc}(e, x) \\ z \leftarrow \mathsf{Eval}(\tilde{f}, \tilde{x}) \end{array} \right] > 1 - \mathrm{negl}(\lambda)$$

*The scheme is **private** if there exists a simulator $\mathcal{S}$ such that for all inputs $x \in \mathcal{X}, y \in \mathcal{Y}$, the following are indistinguishable in λ:*

$$\left\{ (\tilde{f}, \tilde{x}) \;\middle|\; \begin{array}{l} (\tilde{f}, e) \leftarrow \mathsf{Garb}(1^\lambda, y) \\ \tilde{x} \leftarrow \mathsf{Enc}(e, x) \end{array} \right\} \approx \left\{ \mathcal{S}(1^\lambda, x, f(x,y)) \right\}$$

*We say the scheme is **statistical** if the above indistinguishability holds even against unbounded adversaries (i.e., the ensembles are statistically close).*

*The scheme is **affine** if $\mathcal{X} \subset R^{|x|}$ for some finite ring R and for fixed input encoding string e, $\mathsf{Enc}(e, x)$ is component-wise affine. The scheme is **projective** if the scheme is affine and $R = \{0, 1\}$.*

As compared to the definition in [IW14], the above provides separate Garb and Enc procedures, where [IW14] has a single encoding step combining the two. The notions are not meaningfully different, and we separate them because in our main application the garbled circuit is conveyed before input labels. Our definition of affine already subsumes what some works call **decomposable** (Enc acting independently on each input component), since we require Enc to be component-wise affine over $R^{|x|}$.

Adaptive security. The above notion of privacy is *selective*: the evaluator's input is chosen independently of the garbled program $\tilde{f}$. Many applications—including our Bitcoin application—separate an *offline phase*, when the garbled program is sent, from an *online phase*, when the evaluator chooses its input. This requires *adaptive* security, where the adversarial evaluator may choose x after having seen $\tilde{f}$. Adaptive security seems hard to achieve from plain-model symmetric-key cryptography [BHR12a, HJO+16, AAK25], but is easy in the random oracle model: encrypt a selectively-secure garbled program under the output of a random oracle applied to a uniform key, and transmit the key in the online phase. This automatically upgrades any selectively-secure garbling scheme, including ours, to an adaptively-secure one; see [BHR12a] for details.

IT-GSs. As mentioned in Section 1, there exist efficient information-theoretic garbling schemes for certain classes of programs. We here describe one such technique, needed for our Bitcoin application:

Theorem 3.1 (IT-GS for Sums of Monomials [CFIK03, IK02, AIK11, IW14])**.** *Let R be a finite ring. Consider a function $f\colon R^{|x|} \times R^t \to R$ of the form*

$$f(x,y) = \sum_{i=1}^{t} \left(y_i \cdot \prod_{j \in S_i} x_j \right),$$

where each $S_i \subseteq [|x|]$ is a multiset i.e. with repeated indices allowed. There exists a statistical affine partial garbling scheme $\mathsf{PGS}_{\mathsf{poly}}$ for f with $\tilde{f} = \emptyset$ and $|\tilde{x}| = \sum_{i=1}^{t} |S_i|$ ring elements.

Theorem 3.1 is a special case of [IW14]'s IT-GS for arithmetic branching programs: because the evaluator knows x in cleartext, she routes herself through the program without any garbled material ($\tilde{f} = \emptyset$), and the encoding size $\sum_i |S_i|$ counts the edges of the corresponding chains.

3.2 Cryptographic Assumption

We prove our garbling scheme secure under the assumption of a circular correlation robust hash (CCRH) function. CCRHs are relatively ubiquitous in the garbled circuit literature, due to their importance to the Free XOR optimization [KS08, CKKZ12]. Roughly, a CCRH produces random-looking output, under the assumption that inputs are keyed by some correlation $\Delta \in \{0,1\}^\lambda$, even when the output of the CCRH is used to encrypt values that involve Δ. We provide the full definition in Appendix A.

3.3 Switch Systems

Switch systems were introduced by [Hea24] as a computational model that captures much of the expressivity of symmetric-key-based garbling. We recall the model and its key capabilities; we extend the framework in Section 6 to suit our constructions. Looking forward, a main contribution of our work is to demonstrate that switch systems are naturally compatible with IT-GS-style garbled labels, and to demonstrate that this compatibility can lead to non-trivial constructions, in particular our projectivization technique.

A switch system is defined over a collection of *wires*, each carrying a value from some ring or group. Rather than specifying a one-directional computation from inputs to outputs—as Boolean circuits do—a switch system specifies local *constraints* on wires. An evaluator satisfies these constraints in whatever order is convenient to the computation, propagating known wire values to unknown ones. This additional flexibility, relative to Boolean circuits, is precisely what enables efficient garbling of non-Boolean operations. The two principal gate types are as follows.

Switches. A switch takes a data wire x and a control wire $\mathsf{ctrl} \in \{0,1\}$ and produces an output data wire y: when $\mathsf{ctrl} = 0$ the switch *closes*, enforcing $x = y$; when $\mathsf{ctrl} = 1$ it imposes no constraint. The evaluator must know ctrl in cleartext, so in a secure switch system all controls must be public or simulatable. In our construction, all control values can be inferred from E's input x. Switch gates contribute *zero communication* when garbled.

Joins. A join $x \bowtie y$ unconditionally enforces $x = y$ (each switch gets a fresh output wire, so equating outputs requires an explicit join). In the original model [Hea24], each join on a wire over a group $\mathbb{G}$ contributes $\lambda \lg |\mathbb{G}|$ bits to the garbled program. The total communication of a garbled switch system is therefore captured by its total *join width*.

Affine gates. Switch systems support affine operations. Following the Free-XOR technique [KS08], these are communication-free.

One-hot scaling. A key example that illustrates the power of switch systems is that of *one-hot scaling* [HK21, Hea24]. A one-hot vector $h \in \{0,1\}^d$ has exactly one entry equal to 1; scaling h by scalar s produces a vector whose i-th entry equals s if $h_i = 1$ and 0 otherwise. Naively, a garbled circuit for this operation would cost $O(d\lambda)$ bits. In a switch system, one attaches to each index i a switch controlled by h_i: when $h_i = 0$, the switch closes and forces the i-th output to 0; the single "active" output is then resolved by one join enforcing that the sum of all outputs equals s. This costs just $O(\lambda)$ bits, regardless of d. One-hot scaling is a central primitive underlying our constructions in Section 6.

Arithmetic garbling. Building on these ideas, [Hea24] used switch systems to garble arithmetic circuits over $\mathbb{Z}_{2^k}$ from circular correlation robust hashes, achieving garbled gate size $O(k\lambda)$ bits per gate, in contrast to the $O(k^2\lambda)$ bits required by schoolbook multiplication. In [Hea24]'s framework, every wire carries a $k\lambda$-length *control-friendly* label of the form $X + x \cdot \Delta$, where $X \in \mathbb{Z}_{2^k}^{\lambda}$ is a per-wire one-time-pad mask and $\Delta \in \{0,1\}^{\lambda}$ is a global Free-XOR correlation. This format is required for switches to hash-translate between labels on their two data wires. Several building-block switch systems from [Hea24]—including bin-to-hot, bin-to-word, and word-to-ring—are used directly in our constructions.

IT-friendly switch systems. Our setting introduces a complication: we need wires that carry IT-GS-style labels—single elements of a finite group serving as one-time-pad encodings—rather than length-λ vectors. Passing such labels through [Hea24]'s framework as-is would force a λ-fold blowup in communication, eliminating the efficiency advantage of IT-GSs. We resolve this by introducing a distinction between *control-friendly* and *non-control-friendly* (plain) wires. Control-friendly wires behave exactly as in [Hea24]; plain wires carry short group-element labels and are permitted wherever switch-control security is not required. A join between plain wires over a group $\mathbb{G}$ costs only $\lg|\mathbb{G}|$ bits—no λ factor—allowing IT-GS labels to flow through the switch system without incurring additional overhead. We also generalize [Hea24]'s $\mathbb{Z}_{2^k}$-specific gate types to *homomorphism gates* and *subgroup gates* that operate over arbitrary finite groups, encompassing the original gates as special cases. We present the full formal treatment of this extended framework and its associated garbling scheme in Section 6.

3.4 Statistical Smudging

Looking forward, our construction achieves arithmetic over $\mathbb{F}_p$ by lifting to a CRT-friendly ring $\mathbb{Z}_M$, with M large enough that arithmetic does not wrap around. The final reduction modulo p is "outsourced" to E, who performs it locally. This reveals to E a value $x + rp \bmod M$, whereas it is only secure to reveal $x \bmod p$: the overflow r cannot be simulated. We therefore use a standard trick: before revealing, add a large random multiple μp, so that E instead learns $x + (r + \mu)p \bmod M$, and μ statistically hides r. We formalize this well-known fact:

Lemma 3.1 (Smudging Lemma)**.** *Let B be a positive integer and let e be an arbitrary random variable supported on $\{0, \ldots, B-1\}$. Let $\mu \in_{\$} \{0, \ldots, B \cdot 2^{\rho} - 1\}$ be chosen uniformly and independently of e. Then the statistical distance between the distributions of μ and $\mu + e$ is at most $2^{-\rho}$.*

Proof. The statistical distance between μ and $\mu + e$ is

$$\mathrm{SD} = \frac{1}{2} \sum_{x=0}^{B \cdot 2^{\rho} + B - 2} \big|\Pr[\mu = x] - \Pr[\mu + e = x]\big|.$$

For $x \in [B-1,\ B \cdot 2^{\rho} - 1]$, the value $x - e_0$ lies in the support of μ for every $e_0 \in \{0, \ldots, B-1\}$, so

$$\Pr[\mu + e = x] = \sum_{e_0} \Pr[e = e_0] \cdot \frac{1}{B \cdot 2^{\rho}} = \frac{1}{B \cdot 2^{\rho}} = \Pr[\mu = x],$$

and these terms contribute zero. The remaining terms lie in the two edge regions $\{0, \ldots, B-2\}$ and $\{B \cdot 2^{\rho}, \ldots, B \cdot 2^{\rho} + B - 2\}$, each of size at most $B - 1$. Each term contributes at most $\frac{1}{B \cdot 2^{\rho}}$ to the sum, so

$$\mathrm{SD} \ \le\ \frac{1}{2} \cdot \frac{2(B-1)}{B \cdot 2^{\rho}} \ <\ 2^{-\rho}. \qquad \square$$

4 Overview

In Section 1 we motivated the need to augment information-theoretic garbling schemes (IT-GSs) with *projectivity*—where E's input to the garbling scheme is encoded bitwise, by choosing one of two per-bit labels. For functions admitting simple IT-GSs, this yields highly efficient constant-round 2PC protocols: the IT-GS is first projectivized, after which 1-out-of-2 OT enables E to obliviously select her input labels.

4.1 Projectivizing Affine Garbling Schemes

Consider a garbling scheme that computes a function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$. Projectivizing f amounts to building a new garbling scheme that instead computes a function $f' : \{0,1\}^{n'} \times \mathcal{Y} \to \mathcal{Z}$ which accepts E's input as n' bits instead of the native representation of $\mathcal{X}$.

Our projectivization technique can be applied to any garbling scheme that encodes E's input in a component-wise affine manner over a ring $\mathbb{Z}_M$ for arbitrary modulus M. Indeed, G's input need not even belong to the same domain as that of E. For simplicity, assume $\mathcal{X} = \mathbb{F}_p^{|\mathbf{x}|}$ where $\mathbf{x}$ is the evaluator's input. The corresponding projectivized f' will then be $f' : \{0,1\}^{n \cdot |\mathbf{x}|} \times \mathcal{Y} \to \mathcal{Z}$, where $n = \lg |\mathbb{F}_p|$: the scheme first reconstructs each field element from its bits, then evaluates f. Assume that the original garbling scheme produced encodings $(\tilde{f}, \tilde{\mathbf{x}})$ where $\tilde{\mathbf{x}}$ consists of S elements of $\mathbb{F}_p$. Our projectivized garbling scheme produces a garbled program $\tilde{f}'$ of size

$$|\tilde{f}'| = |\tilde{f}| + O((|\mathbf{x}|\lambda + S) \lg p)$$

bits. Concretely, the additive term comes to $\approx 110\,\text{KiB}$ of one-time conversion per input element, plus 593 bits for each encoding element (Section 8). Compared to the bit decomposition approach described in Section 1, our saving is a factor of $\lg p$ whenever $S = \Omega(|\mathbf{x}|\lambda)$. This garbling scheme can be viewed as a method to convert Yao-style projective encodings to IT encodings that are affine over some field $\mathbb{F}_p$. The additive cost we pay for this conversion is, in fact, *proportional to the size of* the input projective encodings and the output affine encodings! Therefore, we describe this conversion as *duty-free*.

4.2 Projectivizing Affine Maps

Recall that for our transformation, we only require that the base garbling scheme for f encodes inputs component-wise via affine functions $\mathbf{a}_i x_i + \mathbf{b}_i$ for $i \in [|\mathbf{x}|]$, where the coefficients $\mathbf{a}_i, \mathbf{b}_i \in \mathbb{F}_p^{S_i}$ depend on G's input and private randomness. Let $S = \sum S_i$. *Projectivizing* such a scheme reduces to *projectivizing affine maps*: if we can efficiently deliver each $\mathbf{a}_i x_i + \mathbf{b}_i$ to an evaluator who holds only bitwise labels for $\mathbf{x}_i = \mathsf{bin}(x_i)$, we are done. With this context, we can appreciate why bit decomposition is so expensive: it expands each x_i into $\lg p$ field-element inputs, and each *bit* then requires its own affine encoding. The encoding grows from S to at least $n \cdot S$ field elements, for Sn^2 bits in total.

4.3 Affine Evaluation via CRT Residues

Consider affine map $\mathbf{a}x + \mathbf{b}$, where $x \in \mathbb{F}_p$ is E's input and $\mathbf{a}, \mathbf{b}$ are private to G. Let x_j denote the j-th bit in x's binary representation. Let $n = \lg p$. Projectivizing $\mathbf{a}x + \mathbf{b}$ involves computing the following:

$$\mathbf{a}\left(\sum_{j=0}^{n-1} 2^j \cdot x_j\right) + \mathbf{b}$$

We garble this map with total communication cost $O(n\lambda + nS)$ bits. The first key idea is to lift this computation from $\mathbb{F}_p$ to a CRT-friendly ring $\mathbb{Z}_M$—a product of small prime fields—and to evaluate the affine map component-wise over each CRT residue. The main challenge is in mapping E's input into such a representation. We next explain our approach to this mapping at a high level. Section 6 later formalizes the technique as a switch system $S_{\mathsf{aff}\text{-}\mathbb{Z}_M}$, and Appendix F provides a concrete cost analysis.

Let $M = \prod_{i=1}^{T} p_i$ be the product of the first T primes, chosen so that $\lg M \geq 2n + \rho$; this ensures that arithmetic over $\mathbb{F}_p$ elements lifts to $\mathbb{Z}_M$ without wrap-around (including a masking step that we explain below). Our construction proceeds in two main stages.

Stage 1: From bits to word chunks. Our first task is to convert the bits of x into a CRT representation. Namely, we must compute $x \bmod p_i$ for each of our CRT primes p_i. The main garbled circuit trick available here is *free addition*. Suppose we choose some big enough (defined later) working modulus 2^ℓ, convert each of the bits x_j to an arithmetic representation modulo 2^ℓ, then linearly compute the following for each i:

$$\left(\sum_{j=0}^{n-1} (2^j \bmod p_i) \cdot x_j\right) \in \mathbb{Z}_{2^\ell}$$

Note that if 2^ℓ is big enough, this is congruent to $x \bmod p_i$. Computing all T such terms requires $O(Tn)$ operations, but because the computation is linear, these operations are "free" in the sense that they do not increase the size of garbling [KS08, Hea24]. The only encoding cost is from casting the bits x_j to *words* over $\mathbb{Z}_{2^\ell}$.

Unfortunately, the above strategy of casting bits one by one—even into the relatively small ring $\mathbb{Z}_{2^\ell}$—is too expensive: it would incur an encoding of size $O(n\ell\lambda)$, and ℓ must be $\Omega(\lg n)$ for the above computation to be correct. This would immediately be our bottleneck. Thus, as a refinement to the above, we partition the n bits of x into $C = \lceil n/\lg n \rceil$ chunks of $\lg n$ bits each. For each chunk x_c, a helper switch system bin-to-word taken from [Hea24] (see Lemma 6.3) *simultaneously* converts all bits of the chunk into a single arithmetic word $w_c \in \mathbb{Z}_{2^\ell}$, amortizing the conversion cost across the chunk. (We cannot afford larger chunks, as the *computation* expended by G/E grows exponentially in chunk size.) Summing over all C chunks, this stage costs $O(n\lambda)$ bits in total.

With our chunks encoded as words, we can again linearly compute values (congruent to) the correct CRT residues. Namely, for each CRT prime p_i, we compute the following:

$$r_i \;:=\; \left(\sum_{c=0}^{C-1} (2^{c \lg n} \bmod p_i) \cdot w_c\right) \;\in\; \mathbb{Z}_{2^\ell}.$$

Stage 2: From CRT residues to affine encodings. For each CRT prime p_i, we now have a value $r_i \in \mathbb{Z}_{2^\ell}$ which is *congruent to*, but not equal to, $x \bmod p_i$. For each prime p_i, we apply a helper switch system word-to-ring (Lemma 6.4) which performs two tasks:

1. Reduces the residue r_i modulo p_i, then
2. Computes $\mathbf{a}x + \mathbf{b} \bmod p_i$ for garbler inputs $\mathbf{a}, \mathbf{b}$.

This computes the i-th component of the CRT decomposition of the affine map:

$$\mathbf{a}x + \mathbf{b} \bmod M \;\cong\; \big(\mathbf{a}x + \mathbf{b} \bmod p_1,\; \ldots,\; \mathbf{a}x + \mathbf{b} \bmod p_T\big)$$

After computing all such residues, E can locally apply the CRT isomorphism to recover $\mathbf{a}x + \mathbf{b} \bmod M$.

Each instance of word-to-ring costs $O(S \lg n)$ bits; summing over $T = \Theta(n/\lg n)$ primes gives $O(Sn)$ bits total. Concretely, this is $\sum_i \lg p_i = 593$ bits per encoding element at our parameters, with no decoding material (Remark 6.1).

This S-dependent cost carries no factor of λ: the output wires carrying the affine encodings are terminal—they are never fed into further switch gates as control inputs, and therefore they do not require the λ-fold label repetition needed for secure evaluation of our hash function. Formalizing this distinction—and the resulting compatibility between arithmetic switch systems and low-overhead IT-GS-style labels—is a main contribution of our approach; see Section 6.

Stage 3: Reducing Modulo p. Now that E holds $\mathbf{a}x + \mathbf{b} \in \mathbb{Z}_M^S$, she can simply locally compute modular reduction to obtain $\mathbf{a}x + \mathbf{b} \bmod p$. Because we set M large enough, this is indeed the correct output. However, as written so far, the approach is insecure: the integers over $\mathbb{Z}_M$ may reveal more about $(\mathbf{a}, \mathbf{b})$ than the same output reduced modulo p. To fix this, G adds a statistical mask $\boldsymbol{\mu} \cdot p$ to $\mathbf{b}$ when lifting to $\mathbb{Z}_M$, where $\boldsymbol{\mu}$ has enough entropy to smudge the integer quotient (Lemma 3.1). The mask vanishes modulo p, so correctness is preserved, and E learns negligible information from $\mathbf{a}x + \mathbf{b} \bmod M$. See Theorem 5.2 for a formal treatment.

5 Projectivizing Garbling Schemes

In this section, we build up to our main result: a recipe for projectivizing any *affine* partial garbling scheme for a function whose (non-private) inputs are elements of a prime field $\mathbb{F}_p$.

5.1 Projective Partial Garbling Scheme for Affine Maps over Prime Fields

We begin by stating our most involved result: a *projective* garbling scheme for computing affine functions over CRT-friendly rings. The construction is formalized as a switch system [Hea24], and we defer its presentation to Section 6. Our remaining results in this section are straightforward applications of this theorem.

Theorem 5.1 (Projective Garbling Scheme for Affine Map over $\mathbb{Z}_M$)**.** *Let* $l\colon \{0,1\}^n \times (\mathbb{Z}_M^S \times \mathbb{Z}_M^S) \to \mathbb{Z}_M^S$ *be defined as*

$$l(\mathbf{x}, \mathbf{a}, \mathbf{b}) := \mathbf{a} \cdot \left(\sum_{j=0}^{n-1} 2^j x_j \right) + \mathbf{b} \bmod M$$

where M *is a primorial larger than* 2^n*. Assuming a CCR hash function, there exists a projective partial garbling scheme* $\mathsf{PGS}_{\mathsf{aff}\text{-}\mathbb{Z}_\mathsf{M}}$ *for* l *with garbled encoding size* $O((\lambda + S) \lg M)$ *bits.*

Proof. Deferred to Section 6. We construct a switch system that computes l, then invoke Theorem 6.1. □

Lifting Affine Maps from $\mathbb{F}_p$ to $\mathbb{Z}_M$. Given a partial garbling scheme for affine maps over $\mathbb{Z}_M$ for a primorial M, we describe how to construct a partial garbling scheme for affine maps over $\mathbb{F}_p$. Throughout, we identify elements of $\mathbb{F}_p$ with their integer representatives in $\{0, \ldots, p-1\}$ and perform arithmetic over $\mathbb{Z}$.

The product $a \cdot x$ lies in $[0, (p-1)^2]$ and adding b gives a value in $[0, p^2)$. Hence, if $M \geq p^2$, no reduction modulo M occurs and

$$(a \cdot x + b) \bmod p = ax + b \in \mathbb{F}_p.$$

In our construction, the garbler adds a statistical mask $\mu \cdot p$ (with μ sampled uniformly from a large domain) that vanishes modulo p but hides the integer quotient $\lfloor (a \cdot x + b)/p \rfloor$, which would otherwise leak the size of a, b. To accommodate both the product and the mask without overflow, we require

$$m := \lg M \geq 2n + \rho,$$

where $n = \lg p$ and ρ is a statistical security parameter.

Theorem 5.2 (Projective Garbling Scheme for Affine Map over $\mathbb{F}_p$)**.** *Let* $l\colon \{0,1\}^n \times (\mathbb{F}_p^S \times \mathbb{F}_p^S) \to \mathbb{F}_p^S$ *be defined as*

$$l(\mathbf{x}, \mathbf{a}, \mathbf{b}) = \mathbf{a} \cdot \left(\sum_{j=0}^{n-1} 2^j x_j \right) + \mathbf{b} \bmod p$$

where p *is a prime modulus and* $n = \lg p$*. Assuming a CCR hash function, there exists a projective partial garbling scheme* $\mathsf{PGS}_{\mathsf{aff}\text{-}\mathbb{F}_\mathsf{p}}$ *for* l *with garbled program size* $O(n(\lambda + S))$ *bits.*

Proof. We construct $\mathsf{PGS}_{\mathsf{aff}\text{-}\mathbb{F}_\mathsf{p}} = (\mathsf{Garb}, \mathsf{Enc}, \mathsf{Eval})$ as follows. Let M, $m := \lg M$, and T be the CRT parameters fixed in Section 2, so that $M \geq p^2 \cdot 2^\rho$. Recall $m = O(n)$ (we assume $\rho < n$). $\mathsf{Garb}(1^\lambda, \mathbf{a}, \mathbf{b})$ interprets $\mathbf{a}, \mathbf{b}$ over $\mathbb{Z}_M$, samples $\boldsymbol{\mu} \in [(M - p^2)/p]^S$ uniformly at random, defines $\mathbf{b}' := \mathbf{b} + \boldsymbol{\mu} \cdot p$, and outputs $(\tilde{f}, e) = \mathsf{Garb}_{\mathsf{aff}\text{-}\mathbb{Z}_\mathsf{M}}(1^\lambda, \mathbf{a}, \mathbf{b}')$. $\mathsf{Enc}(e, \mathbf{x})$ outputs $\mathsf{Enc}_{\mathsf{aff}\text{-}\mathbb{Z}_\mathsf{M}}(e, \mathbf{x})$. $\mathsf{Eval}(\tilde{f}, \tilde{\mathbf{x}})$ computes $\mathbf{z}' = \mathsf{Eval}_{\mathsf{aff}\text{-}\mathbb{Z}_\mathsf{M}}(\tilde{f}, \tilde{\mathbf{x}})$ and outputs $\mathbf{z} := \mathbf{z}' \bmod p \in \mathbb{F}_p^S$. Correctness follows immediately after observing there is no wrap-around modulo M. For privacy, note that $\boldsymbol{\mu}$ is sampled with enough entropy such that $\mathbf{z}'$ can be simulated from the output $\mathbf{z} \in \mathbb{F}_p^S$ by independently sampling $\boldsymbol{\mu} \in [(M - p^2)/p]^S$ and computing $\mathbf{z}' = \mathbf{z} + \boldsymbol{\mu} \cdot p$. This is statistically close to the real distribution of $\mathbf{z}'$ via the Smudging Lemma (Lemma 3.1). □

5.2 From Affine Encodings to Projectivity

With a projective garbling scheme for individual affine maps over $\mathbb{F}_p$ in hand, we now show how to use it as a generic transformation: any partial garbling scheme whose input encoding is component-wise affine over $\mathbb{F}_p$ can be upgraded to a projective scheme at a modest additive cost. The idea is simple—apply $\mathsf{PGS}_{\mathsf{aff}\text{-}\mathbb{F}_\mathsf{p}}$ independently to each input component, converting every affine encoding $x_i \mapsto \mathbf{a}_i \cdot x_i + \mathbf{b}_i$ into a projective one that operates on the bits of x_i.

Corollary 5.1 (Projectivizing Affine Garbling Schemes)**.** *Let $f\colon \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ be a function, where $\mathcal{X} = \mathbb{F}_p^{|x|}$. Suppose there exists an affine partial garbling scheme $\mathsf{PGS} = (\mathsf{Garb}, \mathsf{Enc}, \mathsf{Eval})$ for f. Specifically, assume*

1. Garb *outputs $(\tilde{f}, e)$, where $e = (\mathbf{a}_i, \mathbf{b}_i)_{i\in[|x|]}$ for $\mathbf{a}_i, \mathbf{b}_i \in \mathbb{F}_p^{S_i}$.*
2. $\mathsf{Enc}(e, x) := (\mathbf{a}_i \cdot x_i + \mathbf{b}_i)_{i\in[|x|]}$.

Set $S := \sum_{i\in[|x|]} S_i$. Define the projective function

$$f'\colon \{0,1\}^{n|x|} \times \mathcal{Y} \to \mathcal{Z}$$

where $n := \lg p$ by first recombining bits of x to field elements via $\mathsf{bin\text{-}to\text{-}\mathbb{F}_p}$ and then computing f. Then, assuming the existence of CCRH, there exists a projective partial garbling scheme PGS' for f' with total communication $|\tilde{f}| + O(n(|x|\lambda + S))$ bits.

Proof. Garble each input component independently. Garb' runs Garb to obtain $(\tilde{f}, e)$ with $e = (\mathbf{a}_i, \mathbf{b}_i)_{i\in[|x|]}$, then applies $\mathsf{Garb}_{\mathsf{aff\text{-}\mathbb{F}_p}}$ (Theorem 5.2) to each pair $(\mathbf{a}_i, \mathbf{b}_i)$, outputting garbled program $(\tilde{f}, \tilde{f}_1, \ldots, \tilde{f}_{|x|})$ and the corresponding encoding strings $(\tilde{e}_1, \ldots, \tilde{e}_{|x|})$. Enc' encodes the bits of each x_i under the i-th instance. Eval' first runs each $\mathsf{Eval}_{\mathsf{aff\text{-}\mathbb{F}_p}}$, recovering $\mathbf{a}_i \cdot x_i + \mathbf{b}_i = \mathsf{Enc}(e, x)_i$, then runs Eval on the recovered encodings. Correctness of PGS yields the result. Privacy follows by a standard hybrid argument over the $|x|$ instances of $\mathsf{PGS}_{\mathsf{aff\text{-}\mathbb{F}_p}}$, invoking the privacy of PGS in the final step. Each instance contributes $O(S_i n + \lambda n)$ bits by Theorem 5.2, giving total communication $|\tilde{f}| + O(n(|x|\lambda + S))$ bits. □

Remark 5.1. *Technically, Corollary 5.1 holds even if $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_{|x|}$ is some subset of $\mathbb{F}_p^{|x|}$ (affine is still well-defined per Definition 1). The projective function f' inherits the domain restriction, and we make no guarantees on correctness or privacy when the bit-decomposed input does not correspond to a valid $x \in \mathcal{X}$. This extension is required in our application to Argo MACs in Section 7.1.*

6 Switch System for Affine Maps

In this section, we prove Theorem 5.1 by constructing a switch system $S_{\mathsf{aff\text{-}\mathbb{Z}_M}}$ for affine maps over $\mathbb{Z}_M$, where M is a primorial (in particular, it is CRT-friendly), and then invoking Theorem 6.1, which shows how to garble a switch system and states associated costs.

6.1 Switch Systems with IT Labels

Our construction of $S_{\mathsf{aff\text{-}\mathbb{Z}_M}}$ builds on the switch system framework of [Hea24], which provides a language for expressing garbled computations over integer rings. The original framework was designed to suit the needs of arithmetic garbling, for which it sufficed to consider wires over rings of the form $\mathbb{Z}_{2^k}$. We observe that the framework generalizes naturally to wires over arbitrary finite groups.

We also introduce a distinction between two wire types: *control-friendly* wires, which serve as inputs to switch gates and therefore require labels with λ-fold repetition to ensure hash security, and plain wires, which merely carry data and admit short one-time-pad labels. This distinction is important: it is precisely what allows IT-GS labels, which are single field elements rather than λ-bit strings, to flow through the system without incurring the λ blowup that would otherwise dominate communication.

Definition 2 (IT-Friendly Switch Systems)**.** *A **switch system** is a system of local constraints on wires (i.e. constrained variables), where each wire w_i holds a value over some finite group $\mathbb{G}_i$. The system is defined in terms of gates. Non-input wires are initially* not set *(i.e., have no value), and as the system runs, wires become* set *according to the rules of each gate. We classify wires into two types: those that are **control-friendly**, and those that are not. Each control-friendly wire has a value in a ring expressible as the direct product of integer rings $\mathbb{Z}_{2^{k_0}} \times \mathbb{Z}_{2^{k_1}} \times \ldots$ for wire-specific k_i; non-control-friendly wires can be over any finite group. The types of gates are as follows:*

- *A **switch** takes as input a control-friendly wire s over $\mathbb{Z}_2$, as well as a wire x (which may or may not be control-friendly) over some finite group $\mathbb{G}$. It outputs data wire $y \in \mathbb{G}$. y is control-friendly iff x is. If the* control wire *s becomes set and has value 0, then the switch establishes constraint $x = y$. We denote a switch by writing:*

$$y := x \Leftarrow s \stackrel{?}{=} 0$$

- *A **join** takes input wires x, y from some finite group $\mathbb{G}$, where either both wires are control-friendly, or neither is. It establishes constraint $x = y$. We denote such a gate by simply writing $x = y$.*

- *A **pair** gate takes input wires $x \in \mathbb{G}_0$ and $y \in \mathbb{G}_1$, where either both wires are control-friendly, or neither is. It outputs wire $z \in \mathbb{G}_0 \times \mathbb{G}_1$ and enforces that $z = (x, y)$. z is control-friendly iff x, y are. We denote such a gate by writing:*

$$z := (x, y).$$

- *A **homomorphism** gate is parameterized by a group homomorphism $\varphi : \mathbb{G}_0 \to \mathbb{G}_1$. It takes as input a wire $x \in \mathbb{G}_0$. The gate outputs $y \in \mathbb{G}_1$. It establishes the constraint $y = \varphi(x)$. y is control-friendly iff x is. We denote such a gate by writing $y := \varphi(x)$.*

- *A **subgroup** gate takes as input a wire $x \in \mathbb{G}_0$ where it can be statically deduced from the switch system structure that x is, in fact, in $\mathbb{G}_1$, a subgroup of $\mathbb{G}_0$; it outputs a wire $y \in \mathbb{G}_1$ (y is control-friendly iff x is), and if x becomes set, the gate sets y to $x \in \mathbb{G}_1$. For compatibility with garbled labels, the gate is parameterized by a function $\varphi : \mathbb{G}_0 \to \mathbb{G}_1$ satisfying the "affine" property $\varphi(x + y) = \varphi(x) + y$ for $x \in \mathbb{G}_0, y \in \mathbb{G}_1$. We denote such a gate by writing $y := \varphi(x)$.*

A switch system may have constant *wires, which hold some arbitrary public constant from an arbitrary group. Constant wires may be control-friendly or non-control-friendly.*

*For convenience, we assume each gate has a unique identifier gid. A switch system has two distinguished input wires, one control-friendly wire from ring R_{in}, and one non-control-friendly from group $\mathbb{G}_{in}$. It has a distinguished non-control-friendly output wire from group $\mathbb{G}_{out}$. We say a system is **legal** if for any input $x \in R_{in} \times \mathbb{G}_{in}$ there exists a unique assignment of each wire to a value that satisfies all gate constraints. Because assignments are unique, a legal switch system P computes a function $P : R_{in} \times \mathbb{G}_{in} \to \mathbb{G}_{out}$. From here on, we only consider legal systems.*

Generalizing [Hea24]. Definition 2 is different from the switch system presentation of [Hea24] in two notable ways:

First, the above definition adds a distinction between *control-friendly* and *non-control-friendly* wires. In [Hea24], all wires are implicitly control-friendly. One could instead run [Hea24] unchanged and convert to IT-GS labels outside the formalism, but this requires proving the outside conversion secure, composing the two schemes, and threading the global correlation Δ through the interface. Extending the formalism yields a single proof (Theorem 6.1). Note that a non-control-friendly wire may never control a switch, as its short label cannot be hashed securely.

Second, [Hea24] provides gates that perform specific operations over rings of form $\mathbb{Z}_{2^k}$, including gates that compute affine operations, gates that compute modular reduction, and gates that perform a restricted form of division. Because our formalism must operate over a broader family of sets, we generalize these notions to *homomorphism* and *subgroup* gates.

Our presentation is a strict generalization, so any switch system written in the [Hea24] formalism is automatically a switch system in ours (over control-friendly wires) as well. [Hea24]'s affine gates are captured by pair and homomorphism gates (addition is a homomorphism $\mathbb{G}^2 \to \mathbb{G}$, and affine shifts use constant wires), and its modular reduction gate is itself a homomorphism. Its *division* gate computes $x/2^c \in \mathbb{Z}_{2^k}$ for a wire $x \in \mathbb{Z}_{2^{k+c}}$ that is statically guaranteed to be a multiple of 2^c, and is captured by a subgroup gate followed by a homomorphism. Indeed, the multiples of 2^c form a size-2^k subgroup, the map φ that zeroes the low c bits satisfies the required affine property $\varphi(a + b) = \varphi(a) + b$ for b in the subgroup, and $2^c y \mapsto y$ is a homomorphism from the subgroup to $\mathbb{Z}_{2^k}$. Hence we can use [Hea24]'s switch system constructions directly.

We note that [Hea24] defined a notion of *oblivious* switch systems, i.e. a switch system with randomized inputs where all switch controls can be simulated. There is no difficulty in adding such a notion to Definition 2; but we do not need it for our setting. In our setting, all switch controls will be dictated by E's input x, so already all switch controls can be simulated trivially.

Switch system cost. The communication cost of garbling a switch system is captured by its *join width*. Intuitively, join gates are the only gates that contribute to program size, since all other gate types allow the evaluator to derive output labels for free.

Definition 3 (Join Width)**.** *Consider a join gate constraining $x = y$ where variables x, y are over a group $\mathbb{G}$. We say that this gate $x = y$ has a* ***join width*** *as follows:*

$$\textit{join-width}(x = y) = \begin{cases} \lambda \lg |\mathbb{G}| & \textit{if } x, y \textit{ are control-friendly} \\ \lg |\mathbb{G}| & \textit{otherwise} \end{cases}$$

For a switch system S, the join width of S is defined as the sum of the join widths of each of its join gates.

We now state the main result of this subsection: any legal switch system in the above framework admits a projective partial garbling scheme with encodings of size essentially the join width. Later, we apply this result to reduce the task of constructing a projective partial garbling scheme for affine maps to instead constructing a switch system for the same task.

In short, with the above switch system model set, the construction largely follows that of [Hea24]. Each wire is encoded by some garbled label. Switches are handled by invoking a hash function that allows/disallows E from learning the difference between labels on the two data wires, and they contribute no communication. Joins are handled by including in the garbled program the difference between labels, which costs $\lambda \lg |\mathbb{G}|$ bits when the joined wires are control-friendly and $\lg |\mathbb{G}|$ bits when they are not. Pair, homomorphism, and subgroup gates are handled homomorphically and are free. An application that requires the cleartext output is additionally sent a decoding mask of $\lg |\mathbb{G}_{out}|$ bits for the output wire, though our applications require none (see Remark 6.1 below). Because the handling of gates is relatively straightforward, we postpone proving the following main theorem to Appendix A.

Theorem 6.1. *Let S be a switch system:*

$$S : R_{in} \times \mathbb{G}_{in} \to \mathbb{G}_{out}$$

Let S have join width w. Assuming a CCRH, there exists a partial garbling scheme for any such system with the following efficiencies:

- *Program encodings are of size $w + \lg |\mathbb{G}_{out}|$ bits.*
- *The input encoding string e consists of λ elements from R_{in}.*

Remark 6.1 (No output decoding)**.** *The $\lg |\mathbb{G}_{out}|$ term in Theorem 6.1 is a decoding mask. E's output-wire label has the form $S(x, y) + X$ for a mask X known to G (see Appendix A), so an application that needs the cleartext output must additionally be sent X. In our applications, the affine encodings instead feed an IT-GS (Theorem 3.1). The standard derivation of such a scheme works backward from the output encoding and treats $\mathbf{b}$ as a free parameter, so delivering one specific $\mathbf{b}$ would cost a correction ciphertext per encoding element. However, the derivation can also start from a fixed $\mathbf{b}$, in our case the value determined by the mask X. G chooses the remaining coefficients consistently, and the parties end up with additive shares of the IT-GS output rather than the output itself. Our applications consume such shares directly, so no decoding or correction material is transmitted, and the tallies of Section 6.3 and Section 8 do not include them.*

The main constructive non-triviality is in how to *use* our new switch system gates to achieve our goal, which we discuss next.

6.2 Helper Switch Systems

With the appropriate definitions in place, we now introduce some helper switch systems that serve as building blocks for $S_{\mathsf{aff}\text{-}\mathbb{Z}_M}$. Each is stated with its join width, which will be tallied in Section 6.3 to establish the overall cost of our construction. Our helper systems make extensive use of *one-hot encodings*, which enable performing arbitrary function evaluation inside a switch system.

Lemma 6.1 (One-Hot Truth Table Evaluation)**.** *Let R denote an arbitrary ring. Let $x \in [d]$ be a value, and let $h = \mathsf{hot}(x) \in R^d$ denote a one-hot encoding of x, written over R. (We reserve n for the bit-length of field elements, and write d for the length of a one-hot encoding.) Let $f : [d] \to R$ denote an arbitrary function, and let $\mathcal{T}(f) \in R^d$ denote its truth table. The following holds:*

$$\left(\sum_{i=1}^{d} \mathcal{T}(f)_i \cdot h_i\right) = f(x)$$

Proof. Immediate: since $h = \mathsf{hot}(x)$, the sum selects $\mathcal{T}(f)_x = f(x)$. □

We now present the helper switch systems. These are straightforward modifications of macros in [Hea24]. The first one scale-hot scales a length-d (control-friendly) one-hot encoding by an arbitrary group element, which may or may not be control-friendly.

Lemma 6.2 (scale-hot)**.** *Let $\mathbb{G}$ be a finite group and let $d \geq 1$. There exists a switch system computing the following function, given that h is a one-hot encoding:*

$$\mathsf{scale\text{-}hot} : \{0,1\}^d \times \mathbb{G} \to \mathbb{G}^d$$
$$\mathsf{scale\text{-}hot}(h, s)_i = \begin{cases} 0 \in \mathbb{G} & \text{if } h_i = 0 \\ s & \text{otherwise} \end{cases}$$

The system has d switches and join-width $\lambda \lg |\mathbb{G}|$ if s is control-friendly, and $\lg |\mathbb{G}|$ otherwise.

Proof. We generate the constraints $y_i := (0 \in \mathbb{G}) \Leftarrow h_i = 0$ for each $i \in [d]$, plus the single join $\sum_i y_i = s$, where y is the output vector. If $h_\alpha = 1$, then all $y_{i \neq \alpha} = 0$, and hence $y_\alpha = s$ by the join constraint, correctly implementing the desired function. The join contributes $\lambda \lg |\mathbb{G}|$ bits if s is control-friendly and $\lg |\mathbb{G}|$ otherwise; the switches contribute nothing to join width. □

Next, we introduce a switch system which converts bits of a short integer to an arithmetic word. We require that the integer be *short* as we use one-hot encodings as a stepping stone.

Lemma 6.3 (bin-to-word)**.** *Let $d, k \geq 1$. There exists a switch system (with all wires control-friendly) computing the following function:*

$$\mathsf{bin\text{-}to\text{-}word} : \{0,1\}^d \to \mathbb{Z}_{2^k}$$
$$\mathsf{bin\text{-}to\text{-}word}(x) = \mathsf{bin}^{-1}(x) \bmod 2^k$$

The system has join-width $(d - 1 + k)\lambda$.

Proof. We build on the switch system $\mathsf{bin\text{-}to\text{-}hot} : \{0,1\}^d \to \{0,1\}^{2^d}$ from [Hea24], which converts a binary string to its one-hot encoding with all wires control-friendly, at join-width $(d-1)\lambda$ and $O(2^d)$ switches. Using bin-to-hot, scale-hot, and Lemma 6.1, we define:

$$\mathsf{bin\text{-}to\text{-}word}(c) = \sum_i \mathcal{T}(\mathsf{id})_i \cdot \left(\mathsf{scale\text{-}hot}(\mathsf{bin\text{-}to\text{-}hot}(c),\, 1 \in \mathbb{Z}_{2^k})\right)_i$$

where $\mathsf{id}(x) = x \in \mathbb{Z}_{2^k}$. Correctness is immediate from Lemma 6.1. The constant $1 \in \mathbb{Z}_{2^k}$ is control-friendly, so scale-hot contributes $k\lambda$ join-width by Lemma 6.2, giving total join-width $(d - 1 + k)\lambda$. Note that this construction is implicit in [Hea24], except we do not need one-time masks here. □

Our third helper addresses the following task: given a word encoding (over a control-friendly wire), evaluate an arbitrary function g, scale it by a value $a \in R$ and add offset $b \in R$.

Lemma 6.4 (word-to-ring)**.** *Let R be a finite ring, let $k \geq 1$, and let $g : \mathbb{Z}_{2^k} \to R$ be an arbitrary function. There exists a switch system computing the following function:*

$$\mathsf{word\text{-}to\text{-}ring}_g : \mathbb{Z}_{2^k} \times (R \times R) \to R$$
$$\mathsf{word\text{-}to\text{-}ring}_g(x, (a, b)) = a \cdot g(x) + b$$

The system has join-width $2(k-1)\lambda + \lg |R|$.

Proof. We make use of the following helper function and switch system from [Hea24]:

$$\mathsf{word\text{-}to\text{-}bin} : \mathbb{Z}_{2^k} \to \{0,1\}^k$$
$$\mathsf{word\text{-}to\text{-}bin}(x) = y \qquad \text{where } \textstyle\sum_i 2^i y_i = x$$
$$\mathsf{word\text{-}to\text{-}hot} : \mathbb{Z}_{2^k} \to \{0,1\}^{2^k}$$
$$\mathsf{word\text{-}to\text{-}hot} = \mathsf{bin\text{-}to\text{-}hot} \circ \mathsf{word\text{-}to\text{-}bin}$$

The system word-to-hot has join-width $2(k-1)\lambda$ under Definition 3 and $O(2^k)$ switches, each with $O(k)$-bit data wires.

To compute $\mathsf{word\text{-}to\text{-}ring}_g$, we observe that it factors as:

$$\mathsf{word\text{-}to\text{-}ring}_g(x, (a, b)) = \mathsf{hot\text{-}to\text{-}ring}_g(\mathsf{word\text{-}to\text{-}hot}(x), (a, b))$$

The system $\mathsf{hot\text{-}to\text{-}ring}_g$ is as follows. Let $h = \mathsf{word\text{-}to\text{-}hot}(x) \in \{0,1\}^{2^k}$. First, apply $\mathsf{scale\text{-}hot}(h, a)$, where the garbler input a arrives on the system's non-control-friendly input wire. By Lemma 6.2, this costs a single non-control-friendly join of width $\lg |R|$, enforcing $\sum_i y_i = a$ over the switch outputs y. Next, a homomorphism gate computes $\sum_i \mathcal{T}(g)_i \cdot y_i$, which equals $a \cdot g(x)$ by Lemma 6.1. A final homomorphism gate adds the garbler input b, since garbler inputs are folded into label masks and contribute nothing (see Appendix A). The material E receives from $\mathsf{hot\text{-}to\text{-}ring}_g$ is exactly the one join difference, so the total join width is $2(k-1)\lambda + \lg |R|$. □

Remark 6.2 (Computational cost of word-to-ring)**.** *In our application of* word-to-ring*, we instantiate k with a working modulus $\ell \approx 3 \lg n$, so each invocation incurs $2^\ell(\ell+1)$ hash evaluations. Summed over $T = \Theta(n/\lg n)$ primes, this gives $\Theta(n^4)$ total hash calls, a substantial cost. In Appendix B, we present a more computationally efficient but restricted variant of* word-to-hot *that suffices for our application and reduces this to $O(n^2)$ hash calls. The price is a larger per-residue join-width, 99λ bits in place of $2(\ell-1)\lambda$ at our parameters. Our implementation and the cost analysis of Appendix F use this variant.*

6.3 Affine Maps over CRT Rings

We now construct a switch system $S_{\mathsf{aff}\text{-}\mathbb{Z}_M}$ that computes affine maps of the form $\mathbf{a}x + \mathbf{b}$ by composing the helper systems from the previous subsection.

Parameters. Recall $M = p_1 \times p_2 \times \cdots \times p_T$ where p_i is the i-th prime, $m := \lg M \geq n$. Set $C := \lceil n/\lg n \rceil$ as the number of chunks, $\ell := \lg(C \cdot n \cdot p_T) = O(\lg m)$ as a working modulus, chosen so that residue accumulation in Step 2 does not overflow $\mathbb{Z}_{2^\ell}$, and $T = O(m/\lg m)$ as the number of prime factors of M.

Step 1: Chunk conversion. Partition the input $x \in \{0,1\}^n$ into C chunks $x_0, \ldots, x_{C-1}$, each of length $\lg n$ bits. For each $c \in [C]$, apply bin-to-word (Lemma 6.3) to obtain a control-friendly wire $w_c \in \mathbb{Z}_{2^\ell}$ carrying $\mathsf{bin}^{-1}(x_c) \bmod 2^\ell$.

Step 2: Free accumulation into CRT residues. For each prime $p_{i\in[T]}$, compute a control-friendly wire $r_i \in \mathbb{Z}_{2^\ell}$ via:

$$r_i \;:=\; \sum_{c=0}^{C-1} (2^{c \lg n} \bmod p_i) \cdot w_c \;\in\; \mathbb{Z}_{2^\ell}$$

Each scalar multiplication (by public constant) and addition is a homomorphism; neither contributes join width. Correctness: since $x = \sum_c \mathsf{bin}^{-1}(x_c) \cdot 2^{c \lg n}$, we have $r_i \equiv x \pmod{p_i}$, and our choice of ℓ ensures no overflow.

Step 3: Residue evaluation. For each prime $p_{i\in[T]}$, apply $\mathsf{word\text{-}to\text{-}ring}_{g_i}$ (Lemma 6.4) to the wire r_i, where:

$$g_i : \mathbb{Z}_{2^\ell} \to \mathbb{Z}_{p_i}^S, \qquad g_i(r) := (r \bmod p_i) \cdot \mathbf{1}_S$$

with garbler inputs $(\mathbf{a} \bmod p_i,\ \mathbf{b} \bmod p_i) \in \mathbb{Z}_{p_i}^S \times \mathbb{Z}_{p_i}^S$, which enter on the system's non-control-friendly input wire. This produces an output wire $o_i \in \mathbb{Z}_{p_i}^S$ carrying $\mathbf{a}x + \mathbf{b} \in \mathbb{Z}_{p_i}^S$.

Output. The system outputs the tuple $(o_1, \ldots, o_T)$ via pair gates. By the CRT isomorphism $\mathbb{Z}_M^S \cong \prod_{i=1}^T \mathbb{Z}_{p_i}^S$, this tuple is exactly the CRT representation of $\mathbf{a} \cdot x + \mathbf{b} \in \mathbb{Z}_M^S$.

6.3.1 Join-width

We now tally the join width of $S_{\mathsf{aff}\text{-}\mathbb{Z}_\mathsf{M}}$, gate-by-gate across the three main steps.

1. Each of the C instances of $\mathsf{bin\text{-}to\text{-}word}$ is invoked with input length $\lg n$ and output ring $\mathbb{Z}_{2^\ell}$, contributing join-width $(\lg n - 1 + \ell)\lambda = O(\lg m \cdot \lambda)$ per instance by Lemma 6.3. Summing over $C = \lceil n / \lg n \rceil$ chunks gives $O(m\lambda)$ total, noting $m \geq n$.
2. Homomorphism gates contribute no join width.
3. Each of the T instances of $\mathsf{word\text{-}to\text{-}ring}_{g_i}$ is invoked with $k = \ell = O(\lg m)$ and $R = \mathbb{Z}_{p_i}^S$, contributing $2(\ell - 1)\lambda + S \lg p_i$ per instance by Lemma 6.4. Summing over $i \in [T]$ gives $O(m\lambda) + S \sum_{i=1}^T \lg p_i = O(m\lambda + Sm)$, where the last equality uses $\sum_i \lg p_i = O(m)$ (each ceiled logarithm exceeds $\log_2 p_i$ by less than one, and $T = O(m / \lg m)$).

Combining all steps, the total join-width is exactly

$$\big[C(\lg n - 1 + \ell) + 2T(\ell - 1)\big]\lambda + S \sum_{i=1}^T \lg p_i \;=\; O(m\lambda + Sm)$$

bits: a fixed cost paid once per input element, plus $\sum_i \lg p_i = O(m)$ bits per encoding element (with the hash-optimized $\mathsf{word\text{-}to\text{-}hot}$ variant, the per-residue term $2(\ell - 1)\lambda$ becomes 99λ, cf. Appendix F).

Theorem 5.1 now follows immediately.

Proof of Theorem 5.1. Apply Theorem 6.1 to $S_{\mathsf{aff}\text{-}\mathbb{Z}_\mathsf{M}}$. The join-width bound from Theorem 6.1 gives garbled program size $O(\lg M(\lambda + S))$ bits. Projectivity holds trivially since the input to the switch system is a control-friendly wire $x \in \{0,1\}^n$. □

7 Applications

We now apply Corollary 5.1 to obtain projective partial garbling schemes for computing: (1) Argo MAC encodings [EL26] and (2) private scalar multiplication over BN curves [GKS+26]. We also formalize the claim in Section 1.2 that our technique immediately implies a non-interactive Minicrypt reduction from large field VOLE to OT. Here we state the constructions and results together with their concrete costs. The full constructions and proofs appear in Appendix D.

7.1 Argo MAC Encoding

Argo MAC [EL26] is an elliptic curve homomorphic MAC scheme where the MAC on a point $P \in \mathbb{G}$ is a vector $\sigma(P) + K \in \mathbb{G}^\kappa$, for a group homomorphism $\sigma : \mathbb{G} \to \mathbb{G}^\kappa$ and key $K \in \mathbb{G}^\kappa$, both uniformly sampled by the garbler. To ensure unforgeability, one must set $\kappa = O(\lambda)$. The group $\mathbb{G} = E(\mathbb{F}_p)$ is a BN elliptic curve group[2], and in our comparisons it is BN254. The *Argo MAC encoding* function f_{argo} maps the affine coordinates of a valid curve point to its MAC:

$$f_{\mathsf{argo}}\big((x,y),(\sigma,K)\big) = \sigma(P) + K \qquad \text{where } P := (x,y) \in E(\mathbb{F}_p)$$

This function is defined only on the subset $V := \{(x,y) \mid (x,y) \in E(\mathbb{F}_p)\} \subset \mathbb{F}_p \times \mathbb{F}_p$ of coordinates of valid points, which Definition 1 and Corollary 5.1 allow. The restriction is not a concern in [EL26]'s application, where it is G who provides the curve point and a separate, cheap mechanism (leaking an auxiliary secret that E can use to punish a cheating G) handles invalid inputs.

Implicit in [EL26]'s scheme is an *affine* information-theoretic partial garbling scheme for f_{argo}: each output coordinate is a low-degree polynomial over $\mathbb{F}_p$ whose private coefficients depend on σ, K, and each polynomial can be privately computed via $\mathsf{PGS}_{\mathsf{poly}}$ (Theorem 3.1), which is *affine*.

Theorem 7.1 (Argo MAC encoding)**.** *There exists an information-theoretic affine partial garbling scheme* $\mathsf{PGS}_{\mathsf{argo}}$ *for computing* f_{argo} *with* $\tilde{f} = \emptyset$ *and* $S = O(\kappa)$ *encodings over* $\mathbb{F}_p$*.*

Proof. Implicit in [EL26], Section 2.4 and Section 3.1. □

Corollary 7.1. *Assuming CCRH, there exists a projective partial garbling scheme for* f'_{argo} *(the bitwise-input variant of* f_{argo}*) with total communication* $O(\lambda^2)$ *bits.*

Proof. Apply Corollary 5.1 to $\mathsf{PGS}_{\mathsf{argo}}$. The input encoding consists of $S = O(\lambda)$ field elements in $\mathbb{F}_p$. With $n := \lg p = O(\lambda)$, this gives us total communication $O(Sn + 2\lambda n) = O(\lambda^2)$ bits. □

This is an $O(\lambda)$ factor improvement over the bit-decomposition approach used by [EL26] which incurs total cost $O(\lambda^3)$ bits. Concretely, their protocol (using bit-decomposition) incurs total communication 6.8 MiB for garbling f'_{argo} over BN254 with $\kappa = \lceil \lg p / \log_2 7 \rceil = 91$ MAC components. Our approach, as measured in Section 8, costs ≈ 298 KiB, a 23× improvement. Appendix F gives an analytic upper bound.

7.2 Embryo: Garbled Scalar Multiplication

BABE [GKS+26] implicitly constructs an affine IT-GS, expressed over $\mathbb{F}_p$, for private scalar multiplication over a BN curve, then uses bit-decomposition to projectivize it. Combined with witness encryption for linear pairing relations [GHK+25], this verifies Groth16 proofs on Bitcoin. In BABE's setting, E provides a curve point π as input, while G inputs a private scalar r that should remain hidden beyond what is implied by the output $r \cdot \pi$. We define the *scalar multiplication with input validation* function $f_{\mathsf{SM}}\big((\pi_x,\pi_y),r\big)$, which outputs $r \cdot \pi$ if $\pi := (\pi_x,\pi_y) \in E(\mathbb{F}_p)$ and $\perp$ otherwise. If E feeds in a point not on the curve, the base IT-GS does not guarantee that r remains hidden. We therefore incorporate the check directly into our construction: G uses a PRF key derived from a secret ν embedded in a curve-check polynomial to mask all scalar-multiplication encodings; E can recover ν—and hence the key—only if her input satisfies the curve equation. This adds only $O(1)$ field elements to the encoding, whereas BABE performs the same check using a Yao-style garbled circuit, incurring unnecessary overhead. The scheme itself follows [GKS+26]'s framework: scalar multiplication reduces to n randomized conditional point additions, each expressible as low-degree polynomials in (π_x,π_y) and computed via $\mathsf{PGS}_{\mathsf{poly}}$ (Theorem 3.1).

Theorem 7.2. *In the random oracle model, assuming* PRF *is a secure pseudorandom function, there exists an* affine partial garbling scheme $\mathsf{PGS}_{\mathsf{SM}}$ *for* f_{SM} *with* $\tilde{f} = \emptyset$ *and* $S = O(n)$ *encodings over* $\mathbb{F}_p$*.*

Proof. Deferred to Appendix D.1, which gives the construction together with its correctness and privacy proofs. □

[2] [EL26] in fact generalizes to subgroups of $\mathbb{G}$; we restrict to $\mathbb{G}$ itself for clarity of presentation. Our techniques lead to similar improvements in the more general setting.

Corollary 7.2. *Assuming CCRH and the existence of a random oracle, there exists a projective partial garbling scheme for f'_{SM} (the bitwise-input variant of f_{SM}) with total communication $O(\lambda^2)$ bits.*

Proof. Apply Corollary 5.1 to $\mathsf{PGS}_{\mathsf{SM}}$. The input encoding has $S = O(n)$ elements over $\mathbb{F}_p$ with $n = O(\lambda)$, giving communication $O(Sn + 2\lambda n) = O(\lambda^2)$ bits. □

Embryo offers an $O(\lambda)$ factor improvement in communication over the bit-decomposition approach used by BABE which incurs total cost $O(\lambda^3)$ bits. Concretely, BABE requires 22.16 MiB of communication for the same computation, compared to our ≈ 442 KiB as measured in Section 8, a 51× improvement (Appendix F gives an analytic upper bound).

7.3 $\mathbb{F}_p$-VOLE Secure Against a Malicious Receiver

Our projective garbling scheme for affine maps over a large prime field (Theorem 5.2) immediately yields a non-interactive protocol for chosen-input vector oblivious linear evaluation (VOLE) over $\mathbb{F}_p$, secure against a malicious receiver. The protocol, in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, is exactly our garbling scheme with OT delivering the input labels: the sender garbles $(\mathbf{a}, \mathbf{b})$ via $\mathsf{Garb}_{\mathsf{aff}\text{-}\mathbb{F}_p}$, one OT per bit of the receiver's scalar x delivers the projective input labels, the sender sends $\tilde{f}$, and the receiver evaluates locally. The receiver sends no further messages after the OT calls, so security against a malicious receiver is trivially implied by the privacy of the garbling scheme.

Theorem 7.3 ($\mathbb{F}_p$-VOLE from OT)**.** *In the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, assuming the existence of a CCRH, the protocol above securely realizes length-S chosen-input VOLE over $\mathbb{F}_p$ against a semi-honest sender and a malicious receiver. The protocol makes $\lg p$ invocations of $\mathcal{F}_{\mathsf{OT}}$ and incurs $O((\lambda + S)\lg p)$ bits of additional communication.*

Proof. Correctness is projectivity plus correctness of $\mathsf{PGS}_{\mathsf{aff}\text{-}\mathbb{F}_p}$ (Theorem 5.2). Semi-honest simulation of the sender is trivial, as his view is his own input. The malicious-receiver simulator extracts x from the OT calls, queries the VOLE functionality, and forges $(\tilde{f}, \tilde{x})$ via the garbling scheme's simulator. See Appendix D.2. □

The marginal cost of generating an additional OLE correlation is one $\mathbb{Z}_M$ element, i.e., roughly $2\lg p$ bits (593 bits at our parameters). Our approach is somewhat reminiscent of the CRT-based VOLE-to-OT reduction of [Kon25], which uses SoftSpokenOT [Roy22] to perform small-field VOLE over various CRT residues. Ours improves over theirs in several ways. First, ours is naturally non-interactive, whereas theirs needs an extra range check proof against a malicious receiver, so their approach cannot be used to build a garbling scheme for our Bitcoin application. Second, ours has lower cost, $O((\lambda + S)\lg p)$ bits against their $O((\lambda \lg p/\lg\lg p + S)\lg p)$. Finally, ours consumes roughly 2× fewer OTs, since our receiver inputs a field element rather than an element of a CRT-friendly ring of more than twice the bit-length. Appendix E surveys further related work on OLE and VOLE [Gil99, CDK+22, BCGI18, BCG+19, IKNP03, DHIM25, CK26].

8 Implementation

We implemented our construction in Rust, together with a bit-decomposition baseline whose communication matches the projectivization step of [EL26, GKS+26]. Both implementations garble *and* evaluate the same workload: S affine maps $\mathbf{a}x + \mathbf{b}$ over the BN254 base field, with E's input delivered as $n = 256$ projective labels. Our implementation is available at `https://github.com/alpenlabs/duty-free-bits`. All measurements are single-threaded on an Apple M1 (2020 MacBook Pro), and reported times are medians.

Straight-line evaluation. Section 6 specifies $S_{\mathsf{aff}\text{-}\mathbb{Z}_M}$ as a switch system, whose evaluator satisfies join constraints in whatever order the revealed labels allow. However, E holds x in cleartext, so the order in which joins become solvable is known in advance. Our implementation therefore unrolls the specification into straight-line code performing the same hashes and the same join subtractions, in a fixed order, with no gate graph and no constraint solver. We formalize the protocol as a switch system for ease of proof (Theorem 6.1), and we implement the hand-rolled version for efficiency.

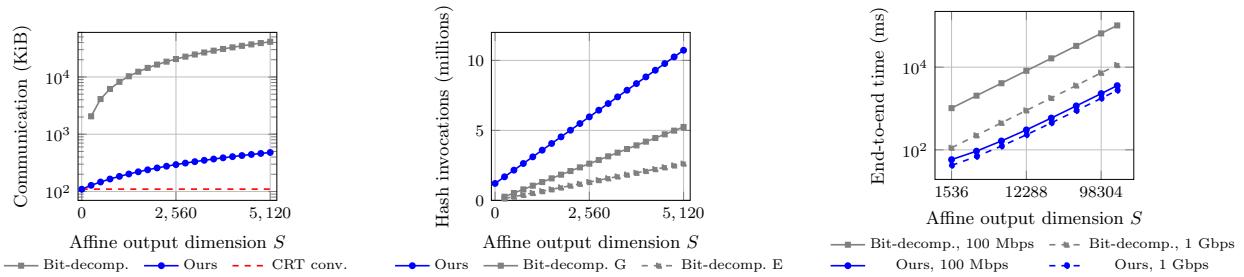

Figure 1: Cost of $S_{\mathsf{aff}\text{-}\mathbb{Z}_M}$ and of the bit-decomposition baseline. Left: communication (log scale). The dashed line marks our constant one-time CRT conversion cost, and the gap above it is exactly $593 \cdot S$ bits of non-control-friendly joins. Middle: CCRH invocations. The baseline's evaluator performs half its garbler's count. Right: end-to-end wall-clock time at 100 Mbps and 1 Gbps (log–log).

Parameters. We fix $\lambda = 128$ and, conservatively, $n = 256$, so that $\lg n = 8$ and the input is processed in $C = 32$ chunks. The CRT basis is the first $T = 80$ primes ($p_{80} = 409$), giving $\sum_{i=1}^{80} \lg p_i = 593$ and $\sum_{i=1}^{80} p_i = 14{,}697$. The working modulus is $\mathbb{Z}_{2^{22}}$: $C \cdot (n-1) \cdot (p_{80} - 1) = 3{,}329{,}280 < 2^{22}$.

Communication. By Theorem 6.1, the garbled program's size splits into a control-friendly (CF) contribution, paid in λ-bit labels, and a non-control-friendly (NCF) contribution, paid in plain bits. Figure 1 (left) plots communication against the affine output dimension S, alongside the bit-decomposition baseline. The CF cost is a constant ≈ 110 KiB, independent of S: the one-time CRT-conversion cost of Steps 1 and 3, shown as the dashed reference line. The NCF cost is exactly $593 \cdot S$ bits at every measured point, i.e. $\sum_{i=1}^{T} \lg p_i$ bits per encoding element, matching the join-width tally of Section 6.3 with no decoding material (Remark 6.1). The baseline transmits $8{,}224 \cdot S$ bytes (one garbled row per bit under garbled row reduction), a factor ≈ 110 more per encoding element. The Embryo computation calls $S_{\mathsf{aff}\text{-}\mathbb{Z}_M}$ once per input coordinate, with $S_x = 5 \cdot 256$ and $S_y = 7 \cdot 256$. The implementation reports a total of ≈ 442 KiB, below the analytic bound of Appendix F.

Hash count. Figure 1 (middle) plots CCRH invocations per party over the same range of S. Where the construction samples elements of $\mathbb{Z}_{p_i}$ from hash output, our implementation does so by rejection sampling, so the reported counts are means over the (input-independent) rejection randomness. Counts apply the standard λ-packing optimization: NCF switches that share a common control bit hash their bundled payload in $\lceil Sw/\lambda \rceil$ blocks, where pads are drawn from w-bit sampling windows ($w = 16$ for $p_i \geq 17$; Appendix F). The CF hash count is a constant ≈ 1.21M, again reflecting the one-time CRT conversion. The NCF hash count, the cost of producing the IT-GS output labels, is $\approx 1{,}859$ calls per encoding element, against the baseline's exact 1,024 (garbler) and 512 (evaluator) calls per element. For the Embryo computation, the implementation reports ≈ 8.1M hash invocations per party.

Runtime. Our goal is to reduce communication, as storage is the bottleneck in our applications. One-hot techniques usually trade communication for computation. However, we find that across a wide range of network settings we improve in runtime as well, because our communication is much smaller. Table 1 reports wall-clock times and hash counts at two workload sizes: $S = 3{,}072$ (the Embryo workload) and $S = 153{,}600$ (50× larger). Our garbler runs at about twice the baseline (2.0× at the Embryo workload, 1.5× at the larger), and our evaluator is $\approx$ 4–5× slower. The evaluator overhead comes from three sources. The one-time Boolean-to-CRT conversion costs a fixed ≈ 1.21M hashes regardless of S, and this cost amortizes as S grows. Producing the output labels costs $\approx 1{,}859$ hash calls per encoding element, compared to the baseline evaluator's 512. Here, a $\approx 2.3\times$ factor is inherent to the algorithm, since our evaluator hashes every slot of each one-hot vector while the baseline evaluator hashes one row per bit, and the rest is the cost of drawing each pad from a 16-bit window by rejection sampling. Finally, ring arithmetic outside the hashes (CRT-encoding $\mathbf{a}, \mathbf{b}$, Garner reconstruction, and the final reduction modulo the field prime) contributes as well.

Table 1: Cost at $n = 256$ (medians; single-threaded Apple M1). E2E = garble + transmit + evaluate.

	$S = 3{,}072$		$S = 153{,}600$	
	ours	baseline	ours	baseline
Garble (ms)	28.4	14.2	1,123	732
Evaluate (ms)	37.8	7.8	1,532	400
Hashes, G (M)	6.92	3.15	286.7	157.3
Hashes, E (M)	6.88	1.57	285.2	78.6
Communication (MB)	0.34	25.3	11.5	1,263
E2E, 100 Mbps (ms)	94	2,043	3,590	102,258
E2E, 1 Gbps (ms)	70	224	2,619	11,304

End-to-end latency. The garbled material is sent in one transmission, so network latency contributes a single round trip regardless of size. We model transmission time as bytes $\cdot$ 8/bandwidth and report garble + transmit + evaluate (Figure 1, right). At 100 Mbps, the baseline needs 2.04 s on the Embryo workload; ours needs 94 ms, a 22$\times$ improvement. At 1 Gbps the gap narrows to 3.2$\times$ (224 ms versus 70 ms), and it widens with S: at $S = 153{,}600$, ours is 28$\times$ faster at 100 Mbps and 4.3$\times$ faster at 1 Gbps. The baseline wins only when S is small enough that its smaller fixed cost dominates, roughly $S \lesssim 270$ at 1 Gbps, far below the workloads of our applications.

Acknowledgements

We thank Robin Linus for pointing out two small bugs (both remedied) in an earlier version of this work. This research was supported by NSF grant CNS-2246353.

References

[AAK25] Anasuya Acharya, Karen Azari, and Chethan Kamath. On the adaptive security of free-XOR-based garbling schemes in the plain model. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VI*, volume 15606 of *LNCS*, pages 214–244. Springer, Cham, May 2025.

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 223–254. Springer, Cham, August 2017.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, October 2011.

[App17] Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. Cryptology ePrint Archive, Report 2017/385, 2017.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BDSW23] Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. Sok: vector OLE-based zero-knowledge protocols. *DCC*, 91(11):3527–3561, 2023.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Berlin, Heidelberg, December 2012.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[CDK+22] Megan Chen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, abhi shelat, and Ran Cohen. Multiparty generation of an RSA modulus. *Journal of Cryptology*, 35(2):12, April 2022.

[CFIK03] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 596–613. Springer, Berlin, Heidelberg, May 2003.

[Che25] Weikeng Chen. SoK: BitVM with succinct on-chain cost. Cryptology ePrint Archive, Paper 2025/1253, 2025.

[CK26] Geoffroy Couteau and Naman Kumar. From ot to ole with almost-linear communication. Cryptology ePrint Archive, Report 2026/295, 2026.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Berlin, Heidelberg, March 2012.

[DHIM25] Jack Doerner, Iftach Haitner, Yuval Ishai, and Nikolaos Makriyannis. From OT to OLE with subquadratic communication. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025*, pages 2249–2263. ACM Press, October 2025.

[Eag25] Liam Eagen. Glock: Garbled locks for bitcoin. Cryptology ePrint Archive, Paper 2025/1485, 2025.

[EL26] Liam Eagen and Ying Tong Lai. Argo MAC: Garbling with elliptic curve MACs. Cryptology ePrint Archive, Paper 2026/049, 2026.

[GHK+25] Sanjam Garg, Mohammad Hajiabadi, Dimitris Kolonelos, Abhiram Kothapalli, and Guru-Vamsi Policharla. A framework for witness encryption from linearly verifiable SNARKs and applications. Cryptology ePrint Archive, Report 2025/1364, 2025.

[Gil99] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, Berlin, Heidelberg, August 1999.

[GKS+26] Sanjam Garg, Dimitris Kolonelos, Mikhail Sergeevitch, Srivatsan Sridhar, and David Tse. BABE: Verifying proofs on Bitcoin made 1000x cheaper. Cryptology ePrint Archive, Report 2026/065, 2026.

[GLNP18] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. *Journal of Cryptology*, 31(3):798–844, July 2018.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

[Hea24] David Heath. Efficient arithmetic in garbled circuits. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 3–31. Springer, Cham, May 2024.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Berlin, Heidelberg, August 2016.

[HK21] David Heath and Vladimir Kolesnikov. One hot garbling. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 574–593. ACM Press, November 2021.

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, November 2000.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan Eidenbenz, and Ricardo Conejo, editors, *ICALP 2002*, volume 2380 of *LNCS*, pages 244–256. Springer, Berlin, Heidelberg, July 2002.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Berlin, Heidelberg, August 2003.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 650–662. Springer, Berlin, Heidelberg, July 2014.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Berlin, Heidelberg, August 2014.

[Kon25] Yashvanth Kondi. Two-party ECDSA signing at constant communication overhead. Cryptology ePrint Archive, Report 2025/1813, 2025.

[KS08] 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 Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Berlin, Heidelberg, July 2008.

[LAA+25] Robin Linus, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, Andrea Pelosi, Orfeas Thyfronitis Litos, Christos Stefo, David Tse, and Alexei Zamyatin. Bridging bitcoin to second layers via BitVM2. Cryptology ePrint Archive, Paper 2025/1158, 2025.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Berlin, Heidelberg, December 2009.

[Roy22] Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 657–687. Springer, Cham, August 2022.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Cham.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Cham, October 2021.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Berlin, Heidelberg, April 2015.

A Garbled Switch System Construction

We start by defining CCR hash functions:

Definition 4 (Circular Correlation Robustness)**.** *Let S denote a set of nonces. Let $H : \{0,1\}^\lambda \times S \to \{0,1\}^\lambda$ be a poly-time computable function. We define the following helper function:*

$$\mathsf{circ} : \{0,1\}^\lambda \times \{0,1\}^\lambda \times S \times \{0,1\} \to \{0,1\}^\lambda$$
$$\mathsf{circ}(\Delta, x, i, b) = H(x \oplus \Delta, i) \oplus b\Delta$$

Consider the following two probability ensembles of oracles:

$$\{\ \mathsf{circ}(\Delta, \cdot, \cdot, \cdot)\ \mid\ \Delta \in_\$ \{0,1\}^\lambda\ \}$$
$$\{\ \mathsf{rand}(\cdot, \cdot, \cdot)\ \mid\ \mathsf{rand} : \{0,1\}^\lambda \times S \times \{0,1\} \to \{0,1\}^\lambda \textit{ is uniform}\ \}$$

*We say a sequence of queries to such an oracle is **legal** if no two queries use the same nonce i. We say that H is **circular correlation robust** if no poly-time adversary issuing legal queries can distinguish the above two ensembles.*

[Hea24] showed that the above definition straightforwardly generalizes to a notion of CCR hashes where the encrypted value Δ is arithmetic over $\mathbb{Z}_{2^k}$ for some k:

Lemma A.1 (CCRH Arithmetic Extension [Hea24])**.** *Let $H : \{0,1\}^\lambda \times S \to \{0,1\}^\lambda$ be a CCR hash function. Then for any (poly-size) k, there exists $H_k : \{0,1\}^\lambda \times S \to \mathbb{Z}_{2^k}^\lambda$ such that H_k is CCR over outputs in $\mathbb{Z}_{2^k}$. Namely, the indistinguishability of Definition 4 holds for the following arithmetic extension of* circ*:*

$$\mathsf{circ}_k : \{0,1\}^\lambda \times \{0,1\}^\lambda \times S \times \mathbb{Z}_{2^k} \to \mathbb{Z}_{2^k}^\lambda$$
$$\mathsf{circ}_k(\Delta, x, i, b) = H_k(x \oplus \Delta, i) + b\Delta$$

In the expression $b\Delta$, each entry of $\Delta \in \{0,1\}^\lambda$ is treated as an element of $\mathbb{Z}_{2^k}$ in the natural manner. H_k issues k black-box calls to H.

Our garbling scheme uses a CCRH H to encrypt group elements from various groups. These encryptions fall into two types:

- We use H to encrypt Yao-style labels holding values over rings that are products of integer rings $\mathbb{Z}_{2^k}$ (every control-friendly wire in our construction is over such a ring). These labels involve a correlation Δ. When we use H in this way, we mean to apply Lemma A.1 coordinate-wise to construct a CCRH that hides Δ.
- We use H to encrypt IT-GS-style labels holding values over some arbitrary finite group. When the output of H is used in this way, we mean to use the randomness from the output of H to uniformly sample a group element, which our implementation realizes by rejection sampling on the hash output.

Proof of Theorem 6.1. The remainder of this section describes the procedures $(\mathsf{Garb}, \mathsf{Enc}, \mathsf{Eval})$ that demonstrate Theorem 6.1, then gives a simulation-based proof that the scheme is private. Note that our scheme is a natural generalization of that of [Hea24], extended to support short, information-theoretic-secure garbled labels.

As is typical in many GC techniques, both Garb and Eval will proceed gate-by-gate. Our scheme arranges the following:

- Recall that each control-friendly wire is over some wire-specific ring R which is a product of integer rings. For each such control-friendly wire $x \in R$, the evaluator will ultimately obtain a value of form $X + x\Delta_R$ on each control-friendly wire x. Here, X is a one-time-pad mask, and Δ_R is some global correlation, akin to a Free XOR correlation [KS08].
- For each non-control-friendly wire $x \in \mathbb{G}$, the evaluator will ultimately obtain a value of form $X + x$, where X is a one-time-pad mask.

Garb first samples a random Free-XOR-style correlation:

$$\Delta \in_\$ \{0,1\}^\lambda$$

Note that we can interpret Δ as being written over the 0 and 1 elements of *any* ring R; we write $\Delta_R \in R^\lambda$ to denote this.

Next, Garb uniformly samples a one-time-pad mask for the evaluator's input wire $x \in R_{in}$:

$$X \in_\$ R_{in}^\lambda$$

Our scheme's input encoding strings e are simply (Δ, X), and the Enc procedure is the following affine map:

$$\mathsf{Enc}((\Delta, X), x) = X + x \cdot \Delta_{R_{in}}$$

Recall that the program also takes as input the garbler's input $y \in \mathbb{G}_{in}$, so we need a mask for this as well. The garbling scheme simply sets the corresponding mask Y as $-y$; this straightforwardly arranges that the evaluator holds $Y + y = 0 \in \mathbb{G}_{in}$, without any communication. This same approach is used to handle constants.

We next describe how each gate type is handled by both Garb and Eval.

Switches.

Consider a switch $y := x \Leftarrow s = 0$. Note that by definition, the wire s is control-friendly and is over the ring $\mathbb{Z}_2$. Thus, it has a mask $S \in \mathbb{Z}_2^\lambda$, and at runtime the evaluator will hold $S + s\Delta \in \mathbb{Z}_2^\lambda$. Let x have mask X. Garb computes a mask on Y as follows:

$$Y := H(S, gid) + X$$

See the discussion of H above. Here gid is the gate's unique identifier. Notice that if $s = 0$, then the evaluator holds S, and hence she can call $H(S, gid)$, obtaining the value $Y - X$, allowing her to translate between values on the x and y wires; that is, if $s = 0$, the evaluator can indeed enforce the constraint $x = y$.

Joins.

Consider a join $x = y$. Let X, Y be the respective one-time-pad masks on these wires. The garbler appends value $X - Y$ to the program encoding $\tilde{f}$. At runtime, this allows the evaluator to enforce constraint $x = y$.

Pair Gates.

Consider a pair gate $z := (x, y)$, and let X, Y be the respective one-time-pad masks on the input wires. The garbler simply defines output wire mask Z as $Z = (X, Y)$; this trivially allows the evaluator to enforce constraint $z = (x, y)$.

Homomorphism Gates.

Consider a homomorphism gate $y := \varphi(x)$. Let X be x's wire mask.

- If x is not control-friendly, the garbler simply defines $Y := \varphi(X)$. Note that because labels are additive, φ is homomorphic over the evaluator's label:

$$\varphi(X + x) = \varphi(X) + \varphi(x) = Y + y$$

  Thus, the evaluator can enforce constraint $y = \varphi(x)$ by simply applying φ to garbled labels.

- If x *is* control-friendly, we extend $\varphi : R \to S$ to $\varphi : R^\lambda \to S^\lambda$ in the natural coordinate-wise manner, and G sets $Y := \varphi(X)$. Similar to above, the evaluator enforces constraint $y = \varphi(x)$ by applying φ to her labels:

$$\varphi(X + x\Delta_R) = \varphi(X) + \varphi(x)\Delta_S = Y + y\Delta_S$$

Subgroup Gates.

Consider a subgroup gate $y := \varphi(x)$. Recall that using this gate requires that it be statically deducible that $x \in \mathbb{G}_0$ is, in fact, in some subgroup $\mathbb{G}_1$ which is the image of φ; ultimately, the gate simply sets $y = x$, but y is in the subgroup. Let X be x's wire mask.

- If x is not control-friendly, the garbler simply defines $Y := \varphi(X)$. Recall that φ satisfies an "affine" property over the subgroup. Thus, φ applies the correct transformation to the evaluator's label:

$$\varphi(X + x \in \mathbb{G}_0) = \varphi(X) + x = Y + y \in \mathbb{G}_1$$

- If x is control-friendly, we extend $\varphi : \mathbb{G}_0 \to \mathbb{G}_1$ to $\varphi : \mathbb{G}_0^\lambda \to \mathbb{G}_1^\lambda$ in the natural coordinate-wise manner, and G sets $Y := \varphi(X)$. E similarly applies φ to her label:

$$\varphi(X + x\Delta_{\mathbb{G}_0}) = \varphi(X) + x\Delta_{\mathbb{G}_1} = Y + y\Delta_{\mathbb{G}_1}$$

Now, recall there is a single output wire $S(x, y) \in \mathbb{G}_{out}$. By the above construction, the evaluator will at runtime hold $S(x, y) + X$ where X is some one-time-pad mask. Thus, the garbler appends X to the program encoding $\tilde{f}$ such that the evaluator can subtract it off and obtain the cleartext output $S(x, y)$. Our applications omit this final mask and consume the masked output directly (Remark 6.1).

We now show the above scheme achieves privacy by constructing a simulator $\mathcal{S}$. Recall that our goal is to demonstrate the following indistinguishability:

$$\left\{ (\tilde{f}, \tilde{x}) \;\middle|\; \begin{array}{l} (\tilde{f}, e) := \mathsf{Garb}(1^\lambda, y) \\ \tilde{x} := \mathsf{Enc}(e, x) \end{array} \right\} \approx \left\{ \mathcal{S}(1^\lambda, x, f(x, y)) \right\}$$

The simulator proceeds as follows. First, it samples an input encoding $\tilde{x}$ uniformly at random from R_{in}^λ. This is a good simulation because in the real garbling, the input encoding is masked by a one-time pad. From here, the simulator proceeds as the garbler would for each gate type, using input wire gate labels to derive output labels. Indeed the handling is identical to Garb for each gate type except the switch.

Consider a switch of form $y := x \Leftarrow s = 0$. By induction, suppose the simulator has already chosen labels S, X respectively for wires s, x. The simulator then proceeds conditionally:

- If the switch has control 0, the simulator's behavior matches the garbler's:

$$Y := H(S, gid) + X$$

  This simulation is consistent, since in the real world evaluation, the evaluator can evaluate the hash function herself.

- If the switch has control 1, the simulator instead samples Y uniformly from the appropriate group.

Choosing output labels from switches uniformly at random is a good simulation due to properties of H. In particular, recall from the discussion above that we use H in two ways:

- For control-friendly data wires, we use the *circular* correlation-robustness of H to encrypt a value that involves Δ.
- For *non*-control-friendly data wires, we use the (non-circular) correlation-robustness of H to generate a uniform group element.

In the real garbling, at a switch with control 1 the evaluator holds the label $L = S + \Delta$, and the switch takes the following form from her perspective:

$$\begin{cases} Y := H(L \oplus \Delta, gid) + X + (y - x)\Delta & \text{if } x, y \text{ are control-friendly} \\ Y := H(L \oplus \Delta, gid) + X + (y - x) & \text{otherwise} \end{cases}$$

That is, the hashed input differs from her label L by the correlation Δ, and the labels X and Y differ by an amount related to their cleartext values. This is the *only* place where Δ appears in the evaluator's view. All other labels/ciphertexts are simply homomorphically derived from labels in $\tilde{x}$ and join gate material included in $\tilde{f}$. Therefore, we can externalize the calls to H that involve Δ to an oracle, then apply the CCRH security property to replace these calls to H by a uniform function. Thus, simulating each label on the output of a switch via a uniform encoding is indeed indistinguishable from the real garbling.

B Hash-Optimized Word-to-Ring Conversions

Recall from Section 6 that each invocation of word-to-ring with working modulus $\ell \approx 3 \lg n$ incurs $2^\ell(\ell+1) = O(n^3 \lg n)$ hash evaluations, giving $\Theta(n^4)$ total across $T = \Theta(n/\lg n)$ primes. This cost is excessive in practice. The culprit is word-to-hot, which must enumerate all 2^ℓ possible values of its input. However, in our application we do not need word-to-ring for an arbitrary function $g : \mathbb{Z}_{2^\ell} \to R$; it suffices to handle functions g that first reduce their input modulo a small prime $p_i = O(n) \ll 2^\ell$. We exploit this structure to replace the single 2^ℓ-way enumeration with a two-phase process that works over much smaller domains, reducing the total hash count to $O(n^2)$ while preserving the same asymptotic communication. Our implementation and the cost analysis of Appendix F use this optimization.

Below, we describe a switch system for converting a word label $r \in \mathbb{Z}_{2^\ell}$ where $\ell \leq 3 \lg n$ to a binary one-hot encoding (OHE) h of $r \bmod p_i$ of length $p_i = O(n)$ with $O(n^2)$ hashes. The construction proceeds in two phases.

Phase 1: Sub-chunk extraction. We decompose the ℓ-bit word r into c sub-chunks by iteratively "peeling off" low-order bits. Let $\ell_0, \ell_1, \ldots, \ell_{c-1}$ be bit-widths summing to ℓ, each at most $\lg n$. In our application, $c = 3$. We process sub-chunks from lowest to highest, maintaining a running remainder r that shrinks at each step.

For $q = 0, 1, \ldots, c-1$:

1. **Extract:** Apply the modular reduction homomorphism to obtain a word label for $r^{(q)} = r \bmod 2^{\ell_q} \in \mathbb{Z}_{2^{\ell_q}}$.
2. **Convert:** Apply a modified word-to-hot gadget to $r^{(q)}$ that simultaneously produces: (a) binary labels for each bit of $r^{(q)}$, (b) an arithmetic OHE of $r^{(q)}$ with entries in $\mathbb{Z}_{2^\ell}^\lambda$, and (c) a $\mathbb{Z}_{2^\ell}$ word label for $r^{(q)}$. Join-width: $(\ell_q - 1 + \ell)\lambda$ bits. See Remark B.1.
3. **Peel:** If $q < c-1$, subtract $r := r - r^{(q)}$ in $\mathbb{Z}_{2^\ell}$ (free); the low ℓ_q bits of r are now zero. Apply free division $r := r/2^{\ell_q}$ (a subgroup gate). The remainder r now encodes the upper $\ell - \ell_0 - \cdots - \ell_q$ bits.

Output (c) is needed for the subtraction in step 3; output (a) is used in Phase 2; output (b) is used to initialize Phase 2 (for $q = 0$).

After Phase 1, we hold: an arithmetic OHE of $r^{(0)}$ (of length 2^{ℓ_0}), and binary labels for all bits of $r^{(0)}, \ldots, r^{(c-1)}$. The total join-width is $\sum_{q=0}^{c-1}(\ell_q - 1 + \ell)\lambda$ bits per prime, and the hash count is $\sum_{q=0}^{c-1} O(2^{\ell_q} \cdot \ell)$ per prime. Since each $\ell_q \leq \lg n$, the per-prime hash count is $O(c \cdot n \cdot \lg n)$, giving $O(n^2)$ total across $T = O(n/\lg n)$ primes.

Remark B.1 (Optimized word-to-hot)**.** *Standard* word-to-hot *[Hea24] internally composes: (1) bin-to-hot on ℓ_q bits, (2) scale by $1 \in \mathbb{Z}_{2^{\ell_q}}$, and (3) solve-bin (free). This yields outputs (a) and (b) above, but with OHE entries in $\mathbb{Z}_{2^{\ell_q}}^{\lambda}$—insufficient for output (c), which requires a $\mathbb{Z}_{2^\ell}$ word label for the subtraction in step 3. Naively obtaining (c) would cost an additional $(\ell_q - 1 + \ell)\lambda$ bits.*

Our modification replaces $1 \in \mathbb{Z}_{2^{\ell_q}}$ with $1 \in \mathbb{Z}_{2^\ell}$ in step (2). The resulting OHE has entries in $\mathbb{Z}_{2^\ell}^{\lambda}$, so hot-to-word immediately yields a $\mathbb{Z}_{2^\ell}$ word label—all three outputs from a single call at cost $(\ell_q - 1 + \ell)\lambda$.

Phase 2: Folding bits into length p OHE. We combine the sub-chunk bits into a binary OHE h of $r \bmod p_i$ of length p_i.

Using Lemma 6.1, we can obtain for free a binary OHE h of $r^{(0)} \bmod p_i$ of length p_i. We then fold in the remaining $\ell_1 + \cdots + \ell_{c-1}$ bits one at a time. Each fold step transforms an OHE h encoding $v \bmod p_i$ into an OHE of $(v + b \cdot 2^j) \bmod p_i$, where b is a binary label at position j:

1. $(s, h') := \mathsf{scale\text{-}hot}(h)$; $s \bowtie b$. Define $(1-b) \cdot h := h - h'$. Cost: p_i hash evaluations, λ bits.
2. $h_{new}[r'] \;:=\; (1-b) \cdot h[r'] \;+\; h'[(r' - 2^j) \bmod p_i]$.

Since addition modulo p_i is commutative, the bits may be folded in any order. After all folds, h encodes $r \bmod p_i$, as required.

The join-width of Phase 2 is $(\ell - \ell_0)\lambda$ bits per prime. The hash count is $(\ell - \ell_0) \cdot p_i$ per prime, giving $(\ell - \ell_0) \cdot \sum_i p_i = O(n^2)$ total.

Total cost. Combining Phases 1 and 2, the per-prime join-width, in units of λ bits, is

$$\underbrace{\textstyle\sum_q (\ell_q - 1 + \ell)}_{\text{Phase 1}} + \underbrace{(\ell - \ell_0)}_{\text{Phase 2}} \;=\; O(\ell),$$

yielding $O(\ell\lambda)$ bits of material per prime and $O(n\lambda)$ total across $T = O(n/\lg n)$ primes. The total hash count is $O(n^2)$, with contributions from both Phase 1 and Phase 2.

C Elliptic Curve Preliminaries

In this section, we provide the background required to understand the application of $\mathsf{PGS}_{\mathsf{aff\text{-}\mathbb{F}_p}}$ to private scalar multiplication over BN curves, as presented in Section 7.2 and Appendix D.

C.1 Notation

We begin by setting up appropriate notation. We follow the notation of [GKS+26] for convenience.

- $E/\mathbb{F}_p$ denotes a BN curve in short Weierstrass form

  $$E: \; y^2 = x^3 + 3 \pmod p,$$

  where p is the BN base-field prime. In our applications, $E/\mathbb{F}_p$ is the BN254 curve.

- $\mathbb{G} = E(\mathbb{F}_p)$ denotes the elliptic curve group, consisting of points that satisfy the above equation, equipped with elliptic curve point addition: given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on E with $P \neq \pm Q$, their sum $P + Q = (x_3, y_3)$ is defined by

  $$\theta = \frac{y_2 - y_1}{x_2 - x_1}, \qquad x_3 = \theta^2 - x_1 - x_2, \qquad y_3 = \theta(x_1 - x_3) - y_1.$$

  For point doubling ($P = Q$), the slope becomes

  $$\theta = \frac{3x_1^2}{2y_1},$$

  and x_3, y_3 are computed by the same formulas. The identity element is the point at infinity $\mathcal{O}$.

- $n := \lg p$. In our application, $n = 254$ (Section 8 and Appendix F conservatively use $n = 256$).
- $\mathbb{F}_q$ denotes the finite field of size $q = |\mathbb{G}|$ i.e. the order of the elliptic curve group. For $\mathbb{G}$ equal to BN254, q is a 254-bit prime.
- We use additive group notation, so for $\pi \in \mathbb{G}$ and $r \in \mathbb{F}_q$, the expression $r \cdot \pi$ denotes the group element obtained by adding π with itself r times.
- For a point $\pi \in \mathbb{G} \setminus \{\mathcal{O}\}$, we write $\pi_x, \pi_y \in \mathbb{F}_p$ for its affine coordinates.
- A *Jacobian representation* of π is a triple $(X, Y, Z) \in \mathbb{F}_p^3$ with $Z \neq 0$ satisfying $\pi_x = X/Z^2$ and $\pi_y = Y/Z^3$. This representation is not unique: (X, Y, Z) and $(\gamma^2 X, \gamma^3 Y, \gamma Z)$ represent the same point for any $\gamma \in \mathbb{F}_p^*$. The point at infinity $\mathcal{O}$ is represented by any $(X, Y, 0)$ with $Y \neq 0$. We write $\mathcal{J}(\pi)$ to denote some Jacobian representation of π (not necessarily canonical).

C.2 Elliptic Curve Addition

We set up lemmas proving that elliptic curve addition can be expressed as a low-degree map over the base field $\mathbb{F}_p$.

Lemma C.1. *Let $\pi = (\pi_x, \pi_y) \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ be given in affine coordinates, and let $\delta \in \{0, 1\}$. Define the map*

$$J_\delta : \mathbb{F}_p^2 \to \mathbb{F}_p^3, \qquad J_\delta(x, y) := \big(1 - \delta + \delta x,\ 1 - \delta + \delta y,\ \delta\big).$$

Then $J_\delta(\pi_x, \pi_y) = \mathcal{J}(\delta\pi)$.

Proof. If $\delta = 1$, then $J_\delta(\pi_x, \pi_y) = (\pi_x, \pi_y, 1)$, i.e., the affine point π embedded into Jacobian coordinates. If $\delta = 0$, then $J_\delta(\pi_x, \pi_y) = (1, 1, 0)$, which represents $\mathcal{O}$ under the convention that any $(X, Y, 0)$ with $Y \neq 0$ encodes the point at infinity. □

Lemma C.2 (Point addition [GKS+26])**.** *Fix $\phi = (\phi_x, \phi_y) \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$. For any input $\pi = (x, y) \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ with $x \neq \phi_x$, define*

$$\begin{aligned} X(x, y) &:= 6 + \phi_x^2\, x - 2\phi_y\, y + \phi_x\, x^2, \\ Y(x, y) &:= 9\phi_y - (\phi_y^2 + 9)\, y + 3\phi_x\phi_y\, x^2 + \phi_y\, y^2 - 3\phi_x^2\, xy, \\ Z(x, y) &:= x - \phi_x. \end{aligned}$$

Then $(X(\pi_x, \pi_y), Y(\pi_x, \pi_y), Z(\pi_x, \pi_y)) = \mathcal{J}(\phi + \pi)$. Moreover, viewed as polynomials in the variables (x, y) (with ϕ_x, ϕ_y fixed), X and Y have total degree 2, and Z has total degree 1.

Lemma C.3 (Conditional point addition)**.** *Fix $\phi = (\phi_x, \phi_y) \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ and let $\delta \in \{0, 1\}$. For any input $\pi = (x, y) \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ with $x \neq \phi_x$, define the polynomial map $J_{\delta,\phi} : \mathbb{F}_p^2 \to \mathbb{F}_p^3$ by*

$$J_{\delta,\phi}(x, y) := \big(X_{\delta,\phi}(x, y),\ Y_{\delta,\phi}(x, y),\ Z_{\delta,\phi}(x, y)\big),$$

where

$$\begin{aligned} X_{\delta,\phi}(x, y) &:= (1 - \delta)\phi_x + \delta\Big(6 + \phi_x^2 x - 2\phi_y y + \phi_x x^2\Big), \\ Y_{\delta,\phi}(x, y) &:= (1 - \delta)\phi_y + \delta\Big(9\phi_y - (\phi_y^2 + 9)y + 3\phi_x\phi_y x^2 + \phi_y y^2 - 3\phi_x^2 xy\Big), \\ Z_{\delta,\phi}(x, y) &:= (1 - \delta) + \delta(x - \phi_x). \end{aligned}$$

Then $J_{\delta,\phi}(\pi_x, \pi_y) = \mathcal{J}(\phi + \delta\pi)$. Moreover, $J_{\delta,\phi}$ has degree 2 as a polynomial map.

Proof. Immediate from Lemmas C.1 and C.2. □

Lemma C.4 (Jacobian randomization [GKS+26])**.** *Given a point $\pi \in E(\mathbb{F}_p)$ in Jacobian coordinates (X, Y, Z), we can obtain a uniform representation of the same point as $\mathrm{diag}(\gamma^2, \gamma^3, \gamma) \cdot (X, Y, Z)^\intercal$ for γ sampled uniformly from $\mathbb{F}_p^*$, where*

$$\mathrm{diag}(\gamma^2, \gamma^3, \gamma) = \begin{pmatrix} \gamma^2 & 0 & 0 \\ 0 & \gamma^3 & 0 \\ 0 & 0 & \gamma \end{pmatrix}.$$

Lemma C.5 (Randomized conditional point addition)**.** *Fix $\phi \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$, $\delta \in \{0, 1\}$, and $\gamma \in \mathbb{F}_p^*$. Define $J_{\delta,\phi,\gamma} \colon \mathbb{F}_p^2 \to \mathbb{F}_p^3$ by*

$$J_{\delta,\phi,\gamma}(x, y) := \mathrm{diag}(\gamma^2, \gamma^3, \gamma) \cdot J_{\delta,\phi}(x, y).$$

Then:

1. *$J_{\delta,\phi,\gamma}(\pi_x, \pi_y) = \mathcal{J}(\phi + \delta\pi)$ for any $\pi = (x, y) \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ with $x \neq \phi_x$.*
2. *If γ is uniform over $\mathbb{F}_p^*$, then $J_{\delta,\phi,\gamma}(\pi_x, \pi_y)$ is uniformly distributed among all Jacobian representations of $\phi + \delta\pi$.*
3. *$J_{\delta,\phi,\gamma}$ has degree 2 as a polynomial map in (x, y).*

Proof. Immediate from Lemmas C.3 and C.4. □

D Application Constructions and Proofs

This appendix gives the constructions and proofs behind the results of Section 7.

D.1 Embryo

Let $H \colon \mathbb{F}_p \to \{0,1\}^\lambda$ be a random oracle and $\mathsf{PRF} \colon \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{F}_p$ a pseudorandom function. Let $\mathbb{G} = E(\mathbb{F}_p)$ be the BN elliptic curve group over $\mathbb{F}_p$, as defined in Appendix C, and let $q := |\mathbb{G}|$. We define the *scalar multiplication with input validation* function $f_{\mathsf{SM}} \colon (\mathbb{F}_p \times \mathbb{F}_p) \times \mathbb{F}_q \to \mathbb{G} \cup \{\perp\}$ as

$$f_{\mathsf{SM}}\big((\pi_x, \pi_y),\, r\big) = \begin{cases} r \cdot \pi & \text{if } \pi := (\pi_x, \pi_y) \in E(\mathbb{F}_p), \\ \perp & \text{otherwise.} \end{cases}$$

That is, the function first checks whether the affine coordinates (π_x, π_y) define a point on the curve, and if so, returns its scalar multiple by r. Following [GKS+26]'s framework, we first present an information-theoretic partial garbling scheme for computing f_{SM}. Applying Corollary 5.1, we obtain a *projective* partial garbling scheme for the function f'_{SM}, modified to accept inputs as bits.

For convenience, we define a program $h(\mathbf{p}, \boldsymbol{\pi})$ which evaluates a bivariate polynomial represented by its coefficient vector $\mathbf{p}$ on input $\boldsymbol{\pi} = (\pi_x, \pi_y) \in \mathbb{F}_p^2$. We denote the result by $\mathbf{p}(\boldsymbol{\pi})$. As a subroutine, we use $\mathsf{PGS}_{\mathsf{poly}} = (\mathsf{Garb}_{\mathsf{poly}}, \mathsf{Enc}_{\mathsf{poly}}, \mathsf{Eval}_{\mathsf{poly}})$, the affine partial garbling scheme for Sum of Monomials from Theorem 3.1 for program h.

We start by describing the garbling procedure. The garbler first samples a secret ν and embeds it into a curve-check polynomial χ, which evaluates to ν if and only if $\boldsymbol{\pi} \in E(\mathbb{F}_p)$ (and some random value otherwise). It derives a key for a PRF $\mathsf{key} = H(\nu)$ and constructs (randomized) conditional-addition maps $J_{i,j}$ encoding the bits of r. Both χ and the $J_{i,j}$ are garbled using $\mathsf{PGS}_{\mathsf{poly}}$; the encoding data of the scalar-multiplication garbling is then masked with PRF outputs keyed by key, so that only an evaluator supplying a valid curve point can recover ν, derive key, and unmask the encodings correctly.

Procedure $\mathsf{Garb}_{\mathsf{SM}}(1^\lambda, r)$:

1. Let $\mathbf{r} = \mathsf{bin}(r)$ denote the bit-decomposition of $r \in \mathbb{F}_q$ i.e. $r = \sum_{i=0}^{n-1} r_i 2^i$ and interpret each bit $r_i \in \{0, 1\} \subset \mathbb{F}_q$ as a field element.

2. Sample $\Delta \in_\$ \mathbb{F}_p^*$ and $\nu \in_\$ \mathbb{F}_p$ uniformly. Define the polynomial

$$\chi(x, y) := (y^2 - x^3 - 3) \cdot \Delta + \nu \in \mathbb{F}_p.$$

Set $\mathsf{key} := H(\nu) \in \{0,1\}^\lambda$ and initialize a counter $\mathsf{idx} := 0 \in \{0,1\}^\lambda$.

3. Sample $\phi_i \in_\$ \mathbb{G}$ uniformly at random subject to $\sum_{i=0}^{n-1} 2^i \phi_i = 0$, and $\gamma_i \in_\$ \mathbb{F}_p^*$ uniformly at random for $i \in [n]$. Define polynomial maps $J_i = J_{r_i,\phi_i,\gamma_i} : \mathbb{F}_p^2 \to \mathbb{F}_p^3$ for $i \in [n]$ as described in Lemma C.5, letting $J_{i,j} : \mathbb{F}_p^2 \to \mathbb{F}_p$ for $j \in \{1,2,3\}$ denote the j-th coordinate map such that $J_{i,j} \in \mathbb{F}_p^{\leq 2}[X, Y]$.

4. Let $\mathbf{p}_{i,j}$ denote the coefficient vector of $J_{i,j}$, and let χ denote the coefficient vector of χ. For each (i, j), compute:

$$\mathsf{Garb}_{\mathsf{poly}}(1^\lambda, \mathbf{p}_{i,j}) = (\mathbf{a}_{i,j,k}, \mathbf{b}_{i,j,k})_{k\in\{x,y\}}, \quad \mathbf{a}_{i,j,k}, \mathbf{b}_{i,j,k} \in \mathbb{F}_p^{S_{i,j,k}}.$$

For each $k \in \{x, y\}$ and each $\ell \in [S_{i,j,k}]$, set

$$\tilde{\mathbf{b}}_{i,j,k}[\ell] := \mathbf{b}_{i,j,k}[\ell] + \mathsf{PRF}(\mathsf{key}, \mathsf{idx}),$$

incrementing idx after each call. Compute $\mathsf{Garb}_{\mathsf{poly}}(1^\lambda, \chi) = (\mathbf{a}_{\chi,k}, \mathbf{b}_{\chi,k})_{k\in\{x,y\}}$.

Vertically concatenate: $\mathbf{a}_k = \{\mathbf{a}_{i,j,k}\}_{i,j} \| \mathbf{a}_{\chi,k}$ and $\tilde{\mathbf{b}}_k = \{\tilde{\mathbf{b}}_{i,j,k}\}_{i,j} \| \mathbf{b}_{\chi,k}$ for $k \in \{x, y\}$, so that $\mathbf{a}_k, \tilde{\mathbf{b}}_k \in \mathbb{F}_p^{S_k}$ where $S_k := \left(\sum_{i,j} S_{i,j,k}\right) + S_{\chi,k}$.

5. Output $\tilde{f} = \emptyset$ and $e = (\mathbf{a}_x, \tilde{\mathbf{b}}_x) \| (\mathbf{a}_y, \tilde{\mathbf{b}}_y)$.

Next, we describe the encoding procedure Enc, which is affine over $\mathbb{F}_p$.

Procedure $\mathsf{Enc}_{\mathsf{SM}}(e, \boldsymbol{\pi})$:

1. Parse $e = (\mathbf{a}_x, \tilde{\mathbf{b}}_x) \| (\mathbf{a}_y, \tilde{\mathbf{b}}_y)$ and $\boldsymbol{\pi} = (\pi_x, \pi_y)$.
2. For $k \in \{x, y\}$, compute $\tilde{\mathbf{c}}_k = \mathbf{a}_k \pi_k + \tilde{\mathbf{b}}_k \in \mathbb{F}_p^{S_k}$.
3. Output $\tilde{\boldsymbol{\pi}} = (\tilde{\mathbf{c}}_x, \tilde{\mathbf{c}}_y)$.

Finally, we describe the evaluation procedure. Using the encoded input, the evaluator first evaluates the curve-check polynomial to recover $\hat{\nu}$ and derive a candidate key for the PRF: $\mathsf{key} = H(\hat{\nu})$. She uses key to unmask the remaining encodings, and evaluates the polynomial-evaluation IT-GS to obtain Jacobian coordinates of points $r_i\pi + \phi_i$. Then, she locally computes a linear combination to recover the output $r\pi$.

For clarity, we use blue to denote the output of each step of $\mathsf{Eval}_{\mathsf{SM}}$, assuming that $(\tilde{f}, \tilde{\boldsymbol{\pi}})$ is correctly generated by composing $\mathsf{Garb}_{\mathsf{SM}}$ and $\mathsf{Enc}_{\mathsf{SM}}$.

Procedure $\mathsf{Eval}_{\mathsf{SM}}(\tilde{f}, \tilde{\boldsymbol{\pi}})$:

1. Parse $\tilde{\boldsymbol{\pi}} = (\tilde{\mathbf{c}}_x, \tilde{\mathbf{c}}_y)$.
2. Parse $\tilde{\mathbf{c}}_k = \{\tilde{\mathbf{c}}_{i,j,k}\}_{i\in[n],j\in[3]} \| \tilde{\mathbf{c}}_{\chi,k}$ for $k \in \{x, y\}$, such that $\tilde{\mathbf{c}}_{i,j,k} \in \mathbb{F}_p^{S_{i,j,k}}$ and $\tilde{\mathbf{c}}_{\chi,k} \in \mathbb{F}_p^{S_{\chi,k}}$.
3. Compute $\hat{\nu} = \mathsf{Eval}_{\mathsf{poly}}(h, \cdot, (\tilde{\mathbf{c}}_{\chi,x}, \tilde{\mathbf{c}}_{\chi,y})) = \chi(\boldsymbol{\pi}) \in \mathbb{F}_p$. Set $\mathsf{key} := H(\hat{\nu}) \in \{0,1\}^\lambda$ and initialize $\mathsf{idx} := 0$.
4. For each (i, j), $k \in \{x, y\}$, and $\ell \in [S_{i,j,k}]$, unmask:

$$\mathbf{c}_{i,j,k}[\ell] := \tilde{\mathbf{c}}_{i,j,k}[\ell] - \mathsf{PRF}(\mathsf{key}, \mathsf{idx}),$$

incrementing idx after each call.

5. For each (i,j), compute $\mathsf{Eval}_{\mathsf{poly}}(h,\cdot,(\mathbf{c}_{i,j,x},\mathbf{c}_{i,j,y})) = v_{i,j} = J_{i,j}(\boldsymbol{\pi}) \in \mathbb{F}_p$.
6. For each $i \in [n]$, interpret $(v_{i,1}, v_{i,2}, v_{i,3}) \in \mathbb{F}_p^3$ as Jacobian coordinates of a point $\psi_i = r_i\pi + \phi_i \in \mathbb{G}$.
7. Compute and output y:

$$y = \sum_{i=0}^{n-1} 2^i\psi_i = (\sum_{i=0}^{n-1} 2^i r_i)\pi + (\sum_{i=0}^{n-1} 2^i\phi_i) = r\pi \in \mathbb{G}$$

Lemma D.1. $\mathsf{PGS}_{\mathsf{SM}} = (\mathsf{Garb}_{\mathsf{SM}}, \mathsf{Enc}_{\mathsf{SM}}, \mathsf{Eval}_{\mathsf{SM}})$ *is* correct.

Proof. Assume $\boldsymbol{\pi} = (\pi_x, \pi_y)$ corresponds to a valid point $\pi \in \mathbb{G}$. Since $\pi \in E(\mathbb{F}_p)$, we have $\chi(\boldsymbol{\pi}) = 0 \cdot \Delta + \nu = \nu$, so $\mathsf{key} = H(\nu)$ and the PRF masks cancel: $\mathbf{c}_{i,j,k} = \mathbf{a}_{i,j,k}\pi_k + \mathbf{b}_{i,j,k}$. By correctness of $\mathsf{PGS}_{\mathsf{poly}}$ (Theorem 3.1), each $v_{i,j} = J_{i,j}(\boldsymbol{\pi}) \in \mathbb{F}_p$. By correctness of $J_{i,j}$ (Lemma C.5), each $\psi_i = r_i\pi + \phi_i \in \mathbb{G}$. Therefore,

$$y = \sum_{i=0}^{n-1} 2^i(r_i\pi + \phi_i) = r \cdot \pi + \sum_{i=0}^{n-1} 2^i\phi_i = r \cdot \pi,$$

where the last equality uses the constraint $\sum_{i=0}^{n-1} 2^i\phi_i = 0$. Note that a semi-honest E holds (π_x, π_y) in the clear so if $\pi \notin \mathbb{G}$ it can output $\perp$ itself, without using the garbled program. □

Lemma D.2. $\mathsf{PGS}_{\mathsf{SM}} = (\mathsf{Garb}_{\mathsf{SM}}, \mathsf{Enc}_{\mathsf{SM}}, \mathsf{Eval}_{\mathsf{SM}})$ *is* private.

Proof. We distinguish two cases based on the validity of $\boldsymbol{\pi}$.

Case 1: $\boldsymbol{\pi} \in E(\mathbb{F}_p)$.

Procedure $\mathcal{S}_{\mathsf{SM}}(1^\lambda, \boldsymbol{\pi}, r \cdot \pi)$:

1. Sample $\nu \in_\$ \mathbb{F}_p$ uniformly. Set $\mathsf{key} := H(\nu)$ and $\mathsf{idx} := 0$.
2. For each $i \in [n]$, sample $\psi_i \in_\$ \mathbb{G}$ uniformly subject to $\sum_{i=0}^{n-1} 2^i\psi_i = r \cdot \pi$. Sample $\gamma_i \in_\$ \mathbb{F}_p^*$ and compute randomized Jacobian coordinates $\psi_i = (v_{i,1}, v_{i,2}, v_{i,3}) \in \mathbb{F}_p^3$.
3. For each (i,j), compute $\mathcal{S}_{\mathsf{poly}}(1^\lambda, h, \boldsymbol{\pi}, v_{i,j}) = \{\mathbf{c}_{i,j,k}\}_{k\in\{x,y\}}$. For each $k \in \{x,y\}$ and $\ell \in [S_{i,j,k}]$, set $\tilde{\mathbf{c}}_{i,j,k}[\ell] := \mathbf{c}_{i,j,k}[\ell] + \mathsf{PRF}(\mathsf{key}, \mathsf{idx})$ and increment idx.
4. Compute $\mathcal{S}_{\mathsf{poly}}(1^\lambda, h, \boldsymbol{\pi}, \nu) = \{\tilde{\mathbf{c}}_{\chi,k}\}_{k\in\{x,y\}}$.
5. Vertically concatenate $\tilde{\mathbf{c}}_k = \{\tilde{\mathbf{c}}_{i,j,k}\}_{i,j} \| \tilde{\mathbf{c}}_{\chi,k}$ for $k \in \{x,y\}$.
6. Output $\tilde{f} = \emptyset$ and $\tilde{\boldsymbol{\pi}} = (\tilde{\mathbf{c}}_x, \tilde{\mathbf{c}}_y)$.

Hyb_1: The real world: $(\tilde{f}, e) \leftarrow \mathsf{Garb}_{\mathsf{SM}}(1^\lambda, r)$, $\tilde{\boldsymbol{\pi}} \leftarrow \mathsf{Enc}_{\mathsf{SM}}(e, \boldsymbol{\pi})$.

Hyb_2: Replace each $\mathsf{Garb}_{\mathsf{poly}}$ with its simulation: for each (i,j), replace generation of $(a_{i,j,k}, b_{i,j,k})$ and input encoding with $\mathcal{S}_{\mathsf{poly}}(1^\lambda, h, \boldsymbol{\pi}, v_{i,j})$ where $v_{i,j} = J_{i,j}(\boldsymbol{\pi})$, applying the same PRF masks. Also replace the curve-check garbling with $\mathcal{S}_{\mathsf{poly}}(1^\lambda, h, \boldsymbol{\pi}, \nu)$. This is indistinguishable from Hyb_1 by privacy of $\mathsf{PGS}_{\mathsf{poly}}$.

Hyb_3: Replace each $\psi_i = r_i\pi + \phi_i$ with $\psi_i \in_\$ \mathbb{G}$ subject to $\sum_{i=0}^{n-1} 2^i\psi_i = r\pi$, and update $v_{i,j}$ used in the above simulations accordingly. This is identically distributed to Hyb_2 since in both hybrids ψ_i are uniform subject to $\sum 2^i\psi_i = r\pi$.

This concludes this case as Hyb_3 is the output of $\mathcal{S}_{\mathsf{SM}}$.

Case 2: $\boldsymbol{\pi} \notin E(\mathbb{F}_p)$.

Procedure $\mathcal{S}_{\mathsf{SM}}(1^\lambda, \boldsymbol{\pi}, \perp)$:

1. Compute $\mathcal{S}_{\mathsf{poly}}(1^\lambda, h, \boldsymbol{\pi}, \chi(\boldsymbol{\pi})) = \{\tilde{\mathbf{c}}_{\chi,k}\}_{k\in\{x,y\}}$.
2. Sample $\tilde{\mathbf{c}}_{i,j,k}[\ell] \in_\$ \mathbb{F}_p$ independently and uniformly for all (i,j,k,ℓ).
3. Vertically concatenate $\tilde{\mathbf{c}}_k = \{\tilde{\mathbf{c}}_{i,j,k}\}_{i,j} \| \tilde{\mathbf{c}}_{\chi,k}$ for $k \in \{x,y\}$.
4. Output $\tilde{f} = \emptyset$ and $\tilde{\boldsymbol{\pi}} = (\tilde{\mathbf{c}}_x, \tilde{\mathbf{c}}_y)$.

Hyb_1: The real world with $\boldsymbol{\pi} \notin E(\mathbb{F}_p)$.

Hyb_2: Replace the curve-check garbling $\mathsf{Garb}_{\mathsf{poly}}$ and input encoding with $\mathcal{S}_{\mathsf{poly}}(1^\lambda, h, \boldsymbol{\pi}, \chi(\boldsymbol{\pi}))$. This is indistinguishable from Hyb_1 by privacy of $\mathsf{PGS}_{\mathsf{poly}}$.

Hyb_3: Replace $H(\nu)$ (used as the PRF key) with a uniformly sampled $\mathsf{key} \in_\$ \{0,1\}^\lambda$. After Hyb_2, the value ν appears in the view only as the input to H. Since $\boldsymbol{\pi} \notin E(\mathbb{F}_p)$, we have $\chi(\boldsymbol{\pi}) = c\Delta + \nu$ where $c := \pi_y^2 - \pi_x^3 - 3 \neq 0$ and $\Delta \in_\$ \mathbb{F}_p^*$; hence $\chi(\boldsymbol{\pi})$ is uniform over $\mathbb{F}_p \setminus \{\nu\}$, statistically independent of ν with error $1/p$. A distinguisher queries H at ν with $\mathsf{negl}(\lambda)$ probability, so this is indistinguishable from Hyb_2.

Hyb_4: Replace each $\mathsf{PRF}(\mathsf{key}, \mathsf{idx})$ with an independent uniform value $\rho_{\mathsf{idx}} \in_\$ \mathbb{F}_p$. Since key is uniform and independent of the adversary's view, this is indistinguishable from Hyb_3 by PRF security.

Hyb_5: Each scalar-multiplication encoding component $\tilde{\mathbf{c}}_{i,j,k}[\ell] = \mathbf{a}_{i,j,k}[\ell]\pi_k + \mathbf{b}_{i,j,k}[\ell] + \rho_{\mathsf{idx}}$ is uniform in $\mathbb{F}_p$ since ρ_{idx} is uniform and independent. Replace each such component with a fresh $\rho'_{i,j,k,\ell} \in_\$ \mathbb{F}_p$. This is identically distributed to Hyb_4.

This concludes this case as Hyb_5 is the output of $\mathcal{S}_{\mathsf{SM}}$. □

Proof of Theorem 7.2. Follows from Lemmas D.1 and D.2. The encoding function $\mathsf{Enc}_{\mathsf{SM}}$ is component-wise affine over $\mathbb{F}_p$ by construction, and $S = S_x + S_y = O(n)$. □

D.2 $\mathbb{F}_p$-VOLE

We work in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model; the relevant ideal functionalities are given below.

Functionality $\mathcal{F}_{\mathsf{OT}}$

Parties. A sender S and a receiver R.
Inputs. S holds $(m_0, m_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$; R holds a choice bit $b \in \{0,1\}$.
Output. R receives m_b. S receives no output.

Functionality $\mathcal{F}^S_{\mathsf{VOLE}}$

Parties. A sender S and a receiver R.
Inputs. S holds $(\mathbf{a}, \mathbf{b}) \in \mathbb{F}_p^S \times \mathbb{F}_p^S$; R chooses a scalar $x \in \mathbb{F}_p$.
Output. R receives $\mathbf{a} \cdot x + \mathbf{b} \in \mathbb{F}_p^S$. S receives no output.

We construct the following protocol Π_{VOLE} in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, using the projective garbling scheme $\mathsf{PGS}_{\mathsf{aff}\text{-}\mathbb{F}_\mathsf{p}} = (\mathsf{Garb}, \mathsf{Enc}, \mathsf{Eval})$ from Theorem 5.2. Here S plays the role of the garbler G and R plays the role of the evaluator E.

Protocol Π_{VOLE} (in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model)

Sender garbles. S runs $(\tilde{f}, e) \leftarrow \mathsf{Garb}(1^\lambda, (\mathbf{a}, \mathbf{b}))$, where the encoding string takes the form $e = (\Delta, X_0, \ldots, X_{\lg p - 1})$ with $\Delta \in \{0,1\}^\lambda$ and each $X_j \in \{0,1\}^\lambda$.

OT for encoding. Write $x = \sum_{j=0}^{\lg p - 1} 2^j x_j \in \mathbb{F}_p$ in binary. For each $j \in [\lg p]$, S and R invoke $\mathcal{F}_{\mathsf{OT}}$: S

inputs $(X_j,\ X_j + \Delta)$ and R inputs choice bit x_j. R receives $X_j + x_j \cdot \Delta$. S sends $\tilde{f}$ to R.

Receiver evaluates. R sets $\tilde{x} := (X_j + x_j\Delta)_{j\in[\lg p]} = \mathsf{Enc}(e,\ x)$ and computes $\mathsf{Eval}(\tilde{f},\ \tilde{x})$.

Observe that the OT step is used precisely to securely implement the Enc procedure of $\mathsf{PGS}_{\mathsf{aff}\text{-}\mathbb{F}_\mathsf{p}}$—each invocation of $\mathcal{F}_{\mathsf{OT}}$ delivers one projective input label to R. After OT, R sends no further messages and evaluates locally; the protocol is therefore non-interactive beyond the OT calls.

Proof of Theorem 7.3. The protocol is correct by projectivity and correctness of $\mathsf{PGS}_{\mathsf{aff}\text{-}\mathbb{F}_\mathsf{p}}$ (Theorem 5.2) because

$$\mathsf{Eval}(\tilde{f}, \mathsf{Enc}(e, x)) = \mathbf{a} \cdot x + \mathbf{b}.$$

Observe that S's view consists solely of his own inputs $(\mathbf{a}, \mathbf{b})$. Formally, semi-honest simulation of S is trivial.

Simulating malicious R is also immediate, since R's only actions are to (1) interact with the OT ideal functionality and (2) receive a garbled encoding, which hides $\mathbf{a}, \mathbf{b}$ by garbling scheme privacy. Formally, the simulator learns R's input x when R tries to invoke the OT functionality. The simulator then sends x to the VOLE functionality and obtains $\mathbf{a}x + \mathbf{b}$. Now, the simulator can use the garbling scheme's simulator to forge a garbling $(\tilde{f}, \tilde{x})$ that outputs $\mathbf{a}x + \mathbf{b}$; it uses $\tilde{x}$ to respond to R's OT queries, and then it sends $\tilde{f}$ to R. The simulator then outputs whatever R outputs. This simulation is indistinguishable from real by privacy of the garbling scheme. □

E Related Work on Vector OLE

OLE was classically solved by the protocol of [Gil99], which computes a secret-shared multiplication ax via the schoolbook algorithm, where one OT is used to compute ax_i for each bit x_i. When values are in a ring $\mathbb{Z}_m$, this basic approach incurs cost $O(\lg^2 m)$ bits per OLE. [CDK+22] used the Chinese Remainder Theorem (CRT) to improve this for rings $\mathbb{Z}_m$ where m is a primorial, at cost $O(\lg m \lg\lg m)$ bits per OLE.

Over the last decade, VOLE greatly improved, thanks to the advent of exciting work on pseudorandom correlation generators (PCGs). PCGs led to random VOLE protocols whose total cost is *sublinear* in the length of the vectors [BCGI18, BCG+19]. While these results are remarkable, they rely on strong cryptographic assumptions, such as variants of Learning Parity with Noise (LPN). This reliance means these techniques are expensive, in the sense that vectors need to be relatively large before asymptotic advantage kicks in, and they also incur high computational cost.

It is thus desirable to also achieve OLE/VOLE from simple and efficient Minicrypt primitives. While such primitives seem insufficient to get strongly sublinear cost, they can be more efficient in some settings. For instance, SoftSpokenOT generalized the classic IKNP OT extension protocol, using Minicrypt primitives to reduce VOLE over small (i.e., poly-sized) fields to OT, and demonstrated best-in-class performance under many network settings [IKNP03, Roy22].

Minicrypt techniques for OLE/VOLE over large fields $\mathbb{F}_p$ have also begun to emerge. A natural goal here is to achieve cost $O(\lg p)$ bits per OLE. [DHIM25] achieved a maliciously-secure reduction from OLE to OT with cost $\tilde{O}(\lg p)$ bits, where $\tilde{O}$ hides $\mathrm{poly}(\lg\lg p)$ factors. Very recently, [CK26] produced a similar reduction from OLE to OT with cost $\lg p \cdot 2^{O(\log^* \lg p)}$ communication and $O((\log^* \lg p)^2)$ rounds, where $\log^*$ denotes the extremely-slow-growing iterated logarithm function.

We compare with the CRT-based reduction of [Kon25] in Section 7.3.

F Concrete Cost Analysis

We derive analytic bounds on the communication and hash count of our scheme, complementing the measurements of Section 8: Embryo (Section 7.2) against BABE [GKS+26], and Argo MAC encodings (Section 7.1) against [EL26]. Parameters are as fixed in Section 8. The communication bounds are conservative, and the totals reported in Section 8 are lower.

Step 1: Chunk Conversion. We apply bin-to-word to each of the $C = 32$ chunks, converting each 8-bit chunk to a control-friendly word in $\mathbb{Z}_{2^{22}}$. Each instance of bin-to-word is invoked with input length $\lg n = 8$ and output ring $\mathbb{Z}_{2^\ell} = \mathbb{Z}_{2^{22}}$, contributing join-width $(\lg n - 1 + \ell)\lambda = 29\lambda$ by Lemma 6.3. Concretely, the $\lg n - 1 = 7$ term comes from bin-to-hot on the 8-bit chunk, and the $\ell = 22$ term comes from scale-hot by $1 \in \mathbb{Z}_{2^{22}}$. Each instance incurs $2^{\lg n} - 2 = 254$ hashes from bin-to-hot and $2^{\lg n} \cdot \ell = 256 \cdot 22 = 5{,}632$ hashes from scale-hot, totaling 5,886 hashes. Summing over all 32 chunks:

$$32 \cdot 29\lambda = 928\lambda \approx 14.5\,\text{KiB communication}, \qquad 32 \cdot 5{,}886 \approx 188\text{K hashes}.$$

Step 2: Free Accumulation into CRT Residues. Step 2 of $S_{\mathsf{aff}\text{-}\mathbb{Z}_M}$ consists entirely of homomorphism gates (constant multiplication and addition on $\mathbb{Z}_{2^\ell}$ labels) and incurs no communication or hashing cost.

Step 3: Residue Evaluation. We apply the hash-optimized word-to-ring$_{g_i}$ (Appendix B) to each of the $T = 80$ residue wires $r_i \in \mathbb{Z}_{2^{22}}$, with $R = \mathbb{Z}_{p_i}^S$. Each invocation proceeds in two phases.

1. **Sub-chunk extraction.** We decompose each 22-bit residue wire r_i into three sub-chunks: low (8 bits), middle (7 bits), and high (7 bits).

   For the low 8 bits, we apply the modified word-to-hot gadget of Appendix B, scaling internally by $1 \in \mathbb{Z}_{2^{22}}$ rather than $1 \in \mathbb{Z}_{2^8}$, at cost:

   $$(8 - 1 + \ell)\lambda = 29\lambda \text{ bits},$$
   $$(2^8 - 2) + 2^8 \cdot 22 = 5{,}886 \text{ hashes}.$$

   We then subtract and free-divide to isolate the middle 7 bits, repeating the gadget at cost $(7-1+\ell)\lambda = 28\lambda$ and 2,942 hashes, and likewise for the high 7 bits. The total cost of this phase per residue is:

   $$(29 + 28 + 28)\lambda = 85\lambda \text{ bits},$$
   $$5{,}886 + 2{,}942 + 2{,}942 = 11{,}770 \text{ hashes}.$$

2. **Folding bits into the OHE.** Now we hold a one-hot encoding of the low 8 bits (of length 256) and binary labels for the remaining 14 middle and high bits. We first reduce the low-bit OHE modulo p_i to obtain a length-p_i OHE of $[r_i]_{p_i}$, then fold in the remaining 14 bits one at a time. Each fold step incurs p_i hashes and λ bits of communication. The total cost of this phase per residue is:

   $$14\lambda \text{ bits},$$
   $$14 \cdot p_i \text{ hashes}.$$

   Summing the hash cost over all 80 primes: $14 \cdot \sum_{i=1}^{80} p_i = 14 \cdot 14{,}697 = 205{,}758$ hashes.

Combining Phases 1 and 2, the cost per residue is $(85 + 14)\lambda = 99\lambda$ communication and $11{,}770 + 14 \cdot p_i$ hashes. Summing over all 80 residues:

$$80 \cdot 99\lambda = 7{,}920\lambda \approx 124\,\text{KiB},$$
$$80 \cdot 11{,}770 + 205{,}758 \approx 1.15\text{M hashes}.$$

The combined ≈ 138.3 KiB and ≈ 1.34M hashes of Steps 1–3 constitute a *one-time CRT-conversion* cost per input element: they scale with the number of inputs being encoded, but *not* with the complexity of the function being evaluated (and therefore the dimension of the affine encodings).

IT-GS labels. The IT-GS computing f_{SM} has encodings of total dimension 12 per conditional point addition: four entries are degree-1 in the coordinates x, y and four are degree-2 (Lemma C.5, counted via Theorem 3.1). Each CRT ring element has size $\sum_{i=1}^{80} \lg p_i = 593$ bits, so with $n = 256$ maps ϕ_i the labels cost $256 \cdot (593 \cdot 12)\,\text{b} \approx 222.5\,\text{KiB}$. The label output uses scale-hot with a non-control-friendly scalar: each prime p_i contributes p_i switches, each drawing D pads from 16-bit sampling windows by rejection, for $D = 12 \cdot 256 = 3{,}072$, i.e., $\lceil D \cdot 16/\lambda \rceil = 384$ expected calls per switch. Summing over all primes gives $14{,}697 \times 384 \approx 5.64$M hashes. Finally, the curve check is an IT-GS of size 5 ring elements, i.e., $593 \cdot 5\,\text{b} \approx 0.5\,\text{KiB}$.

Totals. The Boolean-to-CRT conversion is paid twice, once per coordinate. For Argo MAC encodings [EL26] over BN254 the methodology is identical (the conversion is unchanged and the polynomial shapes are the same), except that $\kappa = \lceil \lg p / \log_2 7 \rceil = 91$ MAC components are evaluated rather than 256 maps. This sets $D = 12 \cdot \kappa = 1{,}092$, i.e., $\lceil 1{,}092 \cdot 16/128 \rceil = 137$ calls per switch and $\kappa \cdot 593 \cdot 12\,\mathrm{b} \approx 79\,\mathrm{KiB}$ of label communication. Neither column includes output-decoding material (Remark 6.1).

	Embryo		Argo MAC	
Component	**Comm.**	**Hashes**	**Comm.**	**Hashes**
CRT conversion ×2	277 KiB	2.68M	277 KiB	2.68M
IT-GS labels	222.5 KiB	5.64M	79.0 KiB	2.01M
Curve check	0.5 KiB	—	—	—
Total	≈ **500** KiB	≈ **8.3**M	≈ **356** KiB	≈ **4.7**M

BABE [GKS+26] requires 22.16 MiB of communication for the scalar multiplication task, and [EL26] report 6.8 MiB for the Argo MAC computation. In comparison, our totals are ≈ 442 KiB and ≈ 298 KiB (Section 8), a 51× and a 23× improvement.