OHMG: One Hot Modular Garbling for Privacy-Free Arithmetic Circuits

Ariel Futoransky[1], Fadi Barbàra[2], Ramses Fernandez[1], Gabriel Larotonda[3,4] and Sergio Demian Lerner[1,5]

[1] Fairgate Labs
[2] Sapienza Università di Roma, Italy
[3] Universidad de Buenos Aires, Argentina
[4] CONICET, Argentina
[5] Rootstock Labs

Abstract. We propose a novel mechanism for garbling wires and gates of a logical circuit in a privacy-free environment, focusing on the authenticity of the protocol. It is based on one-hot encodings, tensor products and elliptic curve arithmetic. This scheme is designed to work with arithmetic gates, but we also show gadgets to implement transitions from binary inputs to arithmetic outputs and vice versa. For our scheme, each arithmetic gate takes at most one ciphertext of material to execute its functionality (assuming knowledge of the garbled inputs and their cleartexts). The security of the scheme is proved in the UC setting.

Keywords: Garbled circuit · ciphertext optimization · one hot product · tensor product

1 Introduction

1.1 Garbled circuits

Garbled circuits [Yao86] (GCs) enable secure multiparty computation (MPC) with minimal interaction. Traditional GC schemes use boolean-gates and require at least one ciphertext per `AND` gate [BHR12b; LP09; Yao86]. With the introduction of Groth16 SNARKs [Gro16] and the recent proliferation of ZKVMs [LLM+24] that output SNARKs, the problem of verifying large computations on garbled circuits has focused on efficiently encoding SNARK verifiers on boolean circuits. This is enough because SNARKs, combined with other recursion techniques, can prove arbitrary computation.

A Groth16 verification requires a high number of modular exponentiations on large integers, which decompose into an even higher number of CPU word multiplications. When using boolean circuits, each word multiplication requires thousands of gates. The end result is that an optimized Groth16 garbled circuit requires about 3 billion non-free gates [Lin25], or over 60 GB of data when garbled; the cruder, unoptimized counts of Table 1 are about 10 billion AND and 32 billion XOR gates. When the circuits must be proven correct by cut-and-choose methods, several of these circuits need to be transmitted and stored until evaluation time.

There are currently at least four optimizations being actively researched:

1. Improving the boolean circuit gadgets that perform arithmetic operations efficiently on the existent garbling schemes.

E-mail: futo@fairgate.io (Ariel Futoransky), ramses.fernandez@fairgate.io (Ramses Fernandez), glaroton@dm.uba.ar (Gabriel Larotonda), sergio@fairgate.io (Sergio Demian Lerner)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-07-22.

2. Replacing the SNARK verifier by an alternate SNARG scheme that requires less computation for SNARK verification, even at the expense of more costly proving.

3. Replacing the SNARK by a Designated Verifier SNARG.

4. Replacing prime-field curves with binary curves whose operations require a lower amount of boolean gates.

In this work, we focus on a fifth approach: designing a garbling scheme that can more naturally represent arithmetic operations commonly required by Prime-field curves, resulting in shorter encodings.

Therefore, in this paper we introduce OHMG, a privacy-free garbling scheme using one-hot encodings, tensor products, and elliptic curve arithmetic over $\mathbb{Z}_p$. It optimizes *arithmetic gates* (for instance, modular addition/multiplication in $\mathbb{Z}_n \to \mathbb{Z}_m$), requiring at most one ciphertext per gate, regardless of input size. Privacy-free means that our approach discloses all wiring states, but prioritizes authenticity: verifiers cannot forge outputs without valid input garblings. On the other side, our scheme can still protect secret output labels; we sketch conditional disclosure of secrets as a use case and leave a formal treatment to future work. The OHMG scheme suits trustless two-party verification settings with dispute resolution, where honest provers are rewarded via timeouts and cheaters are penalized.

Table 1: Gate count for a real Groth16 verifier (BN254 curve). Excludes possible optimizations (e.g. shared subcircuits). Boolean counts use $\sim$ 500 XOR/add (ripple carry) and $\sim$ 6500 AND +20000 XOR/mult (Karatsuba). Sources: [Bar25; Axi24; Con24].

Description	**Arith. Adds**	**Arith. Mults**	**Bool. XORs**	**Bool. ANDs (non-free)**
Full Groth16 BN254 verifier (1 pairing + scalar checks + overhead)	$\sim$ 2,250,000	$\sim$ 1,500,000	$\sim$ 32 billion	$\sim$ 10 billion

Remark 1. The dominant cost is the garbled-circuit material. OHMG uses one ciphertext per addition gate (large integers up to 512 bits) and one per multiplication gate (operands up to 31 bits, per Remark 25), totaling $\sim$100 million ciphertexts. The standard Boolean approach (Yao + half-gate optimizations) uses one ciphertext per AND gate, totaling $\sim$10 billion ciphertexts.

Our new garbling scheme introduces several important benefits over traditional garbled-circuit constructions. In essence, it replaces Boolean-gate-level garbling with an arithmetic, one-hot, elliptic-curve–based representation that is privacy-free but authenticity-preserving. In terms of circuit garbling material, there is a significant improvement if the circuit is computing a function which is essentially arithmetic to begin with, like a Groth16 verification, simplified to the computation of just one pairing of elliptic curves points in the curve family BN254, see Table 1 for a gate count and Table 1 for a ciphertext count, which shows roughly a 100x reduction in circuit material (transmission and storage) using OHMG. Other benefits of the OHMG garbling scheme are:

1. Optimized for Privacy-Free Applications: OHMG is designed for scenarios where privacy is not required, enabling aggressive efficiency optimizations without compromising correctness.

2. Multi-Format Wire Support: it supports Boolean (B), Arithmetic (A), and One-Hot-Encoding (O) wire formats, with seamless and efficient conversions

- Free conversions: $B \to A$, $O \to A$, $O \to B$.
- Single-ciphertext conversions: $B \to O$, $A \to O$, $A \to B$.

3. Zero-ciphertext One-Hot computations: any function operating on a single One-Hot wire can be evaluated for free, requiring no ciphertext.

4. Constant-Cost Multi-Wire Computations with a Tunable Trade-off: any number of functions computed from the same set of two or more wires requires a single ciphertext, independent of gate count; the local evaluation cost is O of the product of the input wire sizes, and the choice of wire sizes and encodings tunes the communication-computation balance per application.

Our main design is based on one-hot encoding ideas of Heath and Kolesnikov [HK21; Hea24], on elliptic curve operations, and on modular arithmetic. One of the main ingredients in our scheme is the computation of the tensor products of these one-hot garblings, which costs only one ciphertext of material regardless the size of the tensor product. In our scheme, the security properties are subsumed to *authenticity* (see the formal definition in Section 2.3 below), a property that states that the verifier/evaluator of the circuit cannot forge a valid output of the garbled scheme, without knowing the garblings of the corresponding inputs of that output. We exhibit a use case for proofs of knowledge in trustless two-party verification settings, where the successful prover receives a reward and a cheater is penalized.

Since there are several ways to implement a garbling scheme, let us give an overview of our intended presentation and cases of use, that justifies our focus on authenticity, instead of privacy. A garbling of a circuit GC involves two parties P, V (or Patrick and Valery):

1. P is the *prover/garbler* of a circuit C on which P, V agree, computing F.

2. V is the *verifier/evaluator* of the circuit C.

3. The gates $\mathcal{G}_i$ of the circuit C are (partially) ordered topologically: that is the i-th gate has no input that is an output of $\mathcal{G}_j$ for $j > i$. A topological order (not necessarily unique) can always be put on an acyclic circuit C.

4. The *input* wires of the circuit C are the wires with no preceding gate, and the *output* wires are those with no subsequent gate.

5. P claims knowledge of certain inputs a that upon evaluation of circuit C, return the output c, that is $F(a) = c$. Patrick's claim is part of a trustless agreement, where Valery needs to confirm the computation's correctness without trusting Patrick.

6. In implementations, the final gate is a Boolean gate verifying that the output of the circuit equals c after evaluating it with the inputs provided by Patrick. Therefore, there is only one output wire, carrying a Boolean variable which returns true or false. The mechanism of resolution is then by negation: if the verifier V obtains the output corresponding to false, then she wins the challenge (the prover's claim is false). Otherwise, after a certain stipulated time-out, the prover's claim is considered to be true, and he wins the challenge.

Remark 2. Circuits are modelled as directed acyclic graphs (DAGs), allowing fan-out: a wire can feed multiple gates without recomputation. Mid-states can thus be output without duplicating subcircuits.

1.2 Comparison with related work

Table 2 summarizes how OHMG relates to prior garbling schemes. Baum [Bau16] provides a foundational decomposition of privacy-free garbling but does not target arithmetic efficiency. Heath-Kolesnikov [HK21] introduces one-hot encodings for efficient gate evaluation with privacy, yet relies on Boolean decomposition for arithmetic operations. AuthOr [BA25] achieves low per-gate cost for Boolean circuits but lacks native arithmetic support. In contrast, OHMG unifies arithmetic, binary, and one-hot wire representations, enabling constant-cost arithmetic gates without bitwidth-dependent blowup. The scheme trades evaluation cost (elliptic-curve operations) for minimal communication (one ciphertext per gate) and provides UC security rather than game-based guarantees alone, properties motivated by trustless verification settings with dispute resolution.

Table 2: Comparison of OHMG with related garbling schemes.

	Baum [Bau16]	**Heath-Kolesnikov [HK21]**	**AuthOr [BA25]**	**OHMG**
Primary goal	Decompose privacy-free vs. private GC	Efficient one-hot garbling, with privacy	Authenticity-oriented Boolean GC	Arithmetic-efficient, privacy-free garbling
Security model	Compositional, context-dependent privacy	Simulation-based, with privacy	Game-based authenticity	UC authenticity (no privacy)
Wire types	Boolean	One-hot + Boolean	Boolean	Arithmetic + binary + one-hot, unified
Arithmetic gates	Via Boolean circuits	Via Boolean decomposition / masking	Via Boolean circuits	Native (uses elliptic-curve arithmetic)
Cost profile	Depends on subcircuit; bitwidth-dependent	Scales with bitwidth	$\sim$1 ciphertext / AND gate; bitwidth-dep.	1 ciphertext / arithmetic gate; bitwidth-independent
Tensor products	No	Implicit	No	Explicit
Bin $\leftrightarrow$ arith. translation	No	Limited	No	Explicit
Target applications	Mixed MPC pipelines	General MPC	Verifiable computation	SNARK verification, trustless dispute resolution

1.3 Why universal composability?

While game-based authenticity proofs suffice for standalone garbling schemes (as reflected in our own Definition 4 in Section 2.3, which captures authenticity without requiring UC) our target setting demands stronger guarantees. The OHMG scheme is designed for trustless two-party verification with dispute resolution, where the garbled circuit does not

operate in isolation. Instead, it interacts with a global trusted-bulletin-board functionality $\mathcal{G}_{\mathsf{trust}}$ (Algorithm 7), which we model as an abstraction of an append-only, publicly readable ledger (Remark 32). This global setup is shared across all protocol sessions and is directly accessible by the environment $\mathcal{Z}$, enabling commitments posted in one session (such as the garbling phase) to be readable in another (such as the verification phase) (Remark 31).

The UC framework naturally handles this cross-session dependency. Furthermore, our security model allows the environment $\mathcal{Z}$ to corrupt parties and read their internal states (Definition 38), adaptively for the stand-alone gate protocols (Remark 41), statically for the commitment-bearing protocols (Remarks 40 and 38), capturing scenarios where an adversary schedules challenges across multiple concurrent verification sessions sharing the same ledger. The UC composition theorem then ensures that security is preserved when OHMG is deployed alongside other protocols sharing the same ledger, a guarantee that game-based proofs cannot provide.

This compositional security, established by Theorem 8, justifies the additional complexity of our UC treatment.

1.4 Notation

For $n \in \mathbb{N}$, we denote $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ the integers modulo n, and $\mathbb{Z}_n^*$ the multiplicative group of invertible elements. We write $a \equiv b \pmod{n}$ if $n \mid (a - b)$. For a function $f : \mathbb{Z}_n \times \mathbb{Z}_m \to \mathbb{Z}_k$, the notation $f(a, b) \pmod{k}$ indicates that the output is reduced modulo k. We identify $\{0, 1\} \simeq \mathbb{Z}_2$. An integer $a \in \mathbb{Z}_n$ with $n = 2^d$ has little-endian binary encoding $\mathsf{bin}(a) = a[0] \cdots a[d-1]$, where $a = \sum_{j=0}^{d-1} a[j] \cdot 2^j$, and $a[0]$ is the least significant bit (LSB). All wires and gates perform modular arithmetic in $\mathbb{Z}_k$ for some $k \leq n$ chosen per wire by the circuit designer; for Boolean operations, $k = 2$. For a vector $V \in \mathbb{Z}_p^n$, we denote its i-th coordinate by V_i. If $V : \mathbb{Z}_n \to \mathbb{Z}_p^k$ is a vector-valued function, then $V(a)_i$ denotes the i-th coordinate of the vector $V(a)$. We write M^T for the transpose of a matrix M, and identify row vectors v with $1 \times n$ matrices, so that v^T is a column vector. For a bit $z \in \{0, 1\}$, we denote its negation by $\overline{z} = \mathsf{NOT}(z) = 1 - z$. A product over an empty index set equals 1, that is: $\prod_{i \in \emptyset} x_i = 1$. A sum over an empty index set equals 0.

2 Definitions, features and security properties

This section provides a review of garbled circuits, focusing on formal definitions, protocol high-level interactions and security properties relevant to our research. We emphasize a scheme prioritizing authenticity over privacy or obliviousness, exploring their applications and related work.

2.1 Elliptic curves

For an elliptic curve E over $\mathbb{Z}_p$, let $G \in E(\mathbb{Z}_p)$ generate a prime-order subgroup $\langle G \rangle$ of order q. We write $[s]G$ for scalar multiplication by $s \in \mathbb{Z}_q$. All scalars and multiplicative masks used by the protocol lie in $\mathbb{Z}_q^*$, and their products and inverses are taken in $\mathbb{Z}_q^*$. When a scalar is displayed as an element of $\mathbb{Z}_p^*$, we use its canonical representative in $\{1, \ldots, q-1\} \subset \mathbb{Z}_p$, but we never perform modular arithmetic mod p on these scalars.

2.2 Formal framework

We begin with formal definitions to establish the foundation of garbled circuits.

Definition 1. A *garbled circuit GC* of a circuit C (computing functionality F) is a consistent garbling of the wires of the circuit C, together with the necessary *material* that allows the evaluator to obtain a garbled output.

Definition 2. A *garbling scheme* is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, where:

- $\mathsf{Gb}(1^\kappa, C) \to (GC, f_e, f_d)$: The garbling algorithm takes a security parameter κ and a circuit C, producing a garbled circuit GC, an encoding function f_e, and a decoding function f_d.
- $\mathsf{En}(f_e, x) \to X$: The encoding algorithm transforms x into a garbled input X.
- $\mathsf{Ev}(GC, X) \to Y$: The evaluation algorithm computes a garbled output Y.
- $\mathsf{De}(f_d, Y) \to y$: The decoding algorithm recovers the function output y in cleartext.
- $\mathsf{ev}(F, x) \to y$: The evaluation algorithm computes the non-garbled output y from x and the function F represented by the circuit C.

Definition 3. A garbling scheme is *correct* if

$$\Pr[\mathsf{De}(f_d, \mathsf{Ev}(GC, \mathsf{En}(f_e, x))) = \mathsf{ev}(F, x)] = 1$$

for all circuits C and inputs x. Since we garble wires rather than gates, we introduce *consistency*: after evaluating a gate with input wires carrying cleartext x, the evaluator obtains output wires evaluated at $f(x)$, where f is the gate functionality. Consistency at the gate level, using one-hot encodings, ensures the circuit's correctness, supporting OHMG's modularity and authenticity-focused design.

2.3 Security properties

In this work we focus on *authenticity* of garbled outputs and on *soundness* against a malicious prover. We do *not* aim to achieve input privacy or obliviousness: in our model the evaluator learns the cleartext input x and only the correctness of the reported output is protected.

For context, standard privacy notions for garbling schemes (indistinguishability-based and simulation-based) can be defined as in Bellare-Hoang-Rogaway [BHR12b]. We recall them only informally and do not prove that OHMG satisfies them.

Definition 4. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ be a garbling scheme. Let us consider the following experiment $\mathsf{Auth}^{\mathcal{G}}_{\mathcal{A}}(1^\kappa)$:

1. The challenger chooses a function F with circuit representation C and an input x (or lets $\mathcal{A}$ choose them). It runs $(GC, f_e, f_d) \leftarrow \mathsf{Gb}(1^\kappa, C)$.
2. It computes the garbled input $X \leftarrow \mathsf{En}(f_e, x)$ and the honest garbled output $Y \leftarrow \mathsf{Ev}(GC, X)$.
3. It gives $(F, C, GC, f_e, f_d, x, X, Y)$ to $\mathcal{A}$, which outputs a string Y'.
4. The experiment outputs 1 if, and only if, $Y' \neq Y$ and $y' := \mathsf{De}(f_d, Y') \notin \{\bot, \mathsf{ev}(F, x)\}$.

We say that $\mathcal{G}$ satisfies *authenticity* if for all PPT adversaries $\mathcal{A}$, the probability that $\mathsf{Auth}^{\mathcal{G}}_{\mathcal{A}}(1^\kappa) = 1$ is negligible in κ. See Corollary 3 for a UC-based realization of this property by our one-hot gate protocol π_f.

Informally, authenticity guarantees that, given a garbled circuit and a garbled input for some x, no PPT evaluator can produce a different garbled output which still decodes to a valid value $y' \neq \mathsf{ev}(F, x)$.

Definition 5. Consider a two-party protocol between a prover P and evaluator V for computing $y = F(x)$. We say the protocol is *sound against a malicious prover* if for every PPT prover $P^\star$ and every environment $\mathcal{Z}$, the output y_V of the honest evaluator V in the real world is either $\perp$ or the correct value $F(x)$, except with negligible probability. See Corollary 6 for a UC-based realization of this property by our cut-and-choose protocol π_{GC}.

We formalize this second property in the UC framework in Section 5, by specifying ideal functionalities $\mathcal{F}_f$ and $\mathcal{F}_{\mathsf{GC}}$ which are sound by construction, and then showing that our protocols π_f and π_{GC} realize them. Authenticity in the sense of Definition 4 is recovered as a corollary of these UC realizations.

Remark 3. In OHMG the evaluator learns the cleartext input x, and we do not protect against leakage about x beyond what follows from knowing x and $F(x)$. We therefore do not claim indistinguishability-based privacy, simulation-based privacy, or obliviousness as defined in [BHR12b]; these are mentioned only for comparison with the classical garbling framework.

2.4 Background and related work

Bellare et al. [BHR12a] extend the garbling framework of [BHR12b] to adaptive security, where inputs may depend on the garbled circuit, with transforms upgrading static-secure schemes; authenticity there prevents malicious evaluators from forging outputs, as in secure outsourcing and one-time programs. Wang et al. [WRK18] optimize authenticated garbling for malicious two-party computation, prioritizing authenticity over privacy. Our authenticity proofs (adaptive corruption at the gate level, Remark 41; static for the commitment-bearing protocols) address the same property in a trustless verification setting.

Baum [Bau16] decomposes SFE circuits so that privacy-free (authenticity-oriented) garbling handles sub-circuits depending on one party's input, reducing overhead for decomposable circuits such as signature verification; our application likewise uses authenticity for output verification in trustless settings. Biçer and Ajorian [BA25] introduce "AuthOr", an authenticity-oriented scheme for Boolean circuits at one ciphertext per AND gate, compatible with FreeXOR [KS08], targeting verifiable computing and ZKPs; Table 2 positions OHMG against these works.

2.5 Our take on commitments and interactions

Denote $\mathbb{Z}_p^*$ as the non-zero elements of $\mathbb{Z}_p$ (where p is prime), and fix a global *correlation* $\Delta \in \mathbb{Z}_p^*$ to mask wire encodings securely. More details can be found in Section 2.6.

1. The circuit C computes F, where the prover claims that if $x = a$ then $F(a) = c$.
2. The input is binary, that is, $x = x[0] \dots x[d-1]$, with $x[d-1]$ the least significant bit of $x \in \mathbb{Z}_n$.
3. Input wires w_i, $i \in I$, of the garbled circuit GC are binary arrays of Boolean wires (Definition 15 in Section 4).
4. The prover selects Δ (Remark 13) and generates labels for all input wires, picking R_i for all $i \in I$, computing $w_i(0) = R_i$, $w_i(1) = \Delta R_i$ (Definition 14), and hashes $h_{i,\sigma} = \mathsf{Hash}(w_i(\sigma))$, where $H : \mathbb{Z}_p \to \{0,1\}^\lambda$.
5. The prover commits to wire labels using hashes and a Trusted Verification Service $\mathcal{G}_{\mathsf{trust}}$ (Algorithm 7), which abstracts an append-only, publicly readable bulletin board.

6. The output $y \in \mathbb{Z}_m$ (that is, $F : \mathbb{Z}_n \to \mathbb{Z}_m$) has two possible garbled forms, defined in the prover-verifier setup (Definition 15), namely: (i) Binary output: GC produces a binary vector of Boolean wires as a binary garbling, and (ii) Arithmetic one-hot output: GC produces a one-hot vector of Boolean wires.

Remark 4. We model this trusted verifying service as an instance of the global bulletin-board functionality $\mathcal{G}_{\mathsf{trust}}$ introduced in Section 5.2 (Algorithm 7). Algorithms 1 and 2 should therefore be read as the application-level protocol run by the prover and verifier *on top of* $\mathcal{G}_{\mathsf{trust}}$.

2.5.1 Commitment and security for OHMG in the UC framework

Algorithm 1 Prover-$\mathcal{G}_{\mathsf{trust}}$ Commitment and Validation Scheme

Require: Circuit C for $F : \mathbb{Z}_n \to \mathbb{Z}_m$, input/output wire sets I, O, hash function $\mathsf{Hash} : \mathbb{Z}_p \to \{0,1\}^\lambda$, statistical security parameter s. Standing: κ, p. Each copy $j \in [s]$ uses an independent correlation $\Delta_j \xleftarrow{\$} \mathbb{Z}_q^*$ and independent garbling randomness; no secret is shared across copies.

Phase 1: commit input/output wire labels.

1: **for** each copy $j \in [s]$ and each wire $w \in I \cup O$ **do**

2: Sample $R_{w,j} \xleftarrow{\$} \mathbb{Z}_q^*$; set $w_j(\sigma) := R_{w,j}\Delta_j^\sigma$ for $\sigma \in \{0,1\}$.

3: Send hashes $h_{w,\sigma,j} := \mathsf{Hash}(w_j(\sigma))$ to $\mathcal{G}_{\mathsf{trust}}$, under keys extended with the copy index j.

4: **end for**

5: $\mathcal{G}_{\mathsf{trust}}$ stores all hashes (indexed by prover identity and C) and returns receipts $c_{\mathsf{input}}, c_{\mathsf{output}}$.

Phase 2: cut-and-choose.

6: P computes, for each $j \in [s]$, the garbling GC_j from the copy-j labels $\{w_j(\sigma)\}$ (with correlation Δ_j), and sends $\{GC_j\}_{j=1}^s$ to V.

7: V samples $J \subset [s]$, $|J| = s/2$, uniformly; for each $j \in J$, P opens GC_j, V validates the revealed labels via $\mathcal{G}_{\mathsf{trust}}$ against the committed hashes, aborting on any mismatch.

Phase 3: validation during evaluation.

8: Upon receiving garbled input g_i for wire i, $\mathcal{G}_{\mathsf{trust}}$ accepts iff $\mathsf{Hash}(g_i)$ matches the stored hash h_{i,z_i} for some $z_i \in \{0,1\}$; otherwise the evaluator aborts.

Remark 5. The evaluator interacts with $\mathcal{G}_{\mathsf{trust}}$ during the evaluation phase (detailed in the Evaluator Interaction Algorithm) to validate garbled inputs and outputs for $j \notin J$, ensuring security against a corrupt garbler. This occurs after cut-and-choose, using cleartext inputs obtained per algorithm De below.

Algorithm 2 Evaluator Interaction with $\mathcal{G}_{\mathsf{trust}}$ for Validation

Require:
Garbled circuits GC_j for $j \notin J$ and receipts c_{input}, c_{output} from prover. Cleartext input $x = x[0] \dots x[d-1]$ obtained from algorithm De. Trusted verification service $\mathcal{G}_{\mathsf{trust}}$ with stored hashes. Subset J from cut-and-choose (validated by $\mathcal{G}_{\mathsf{trust}}$).

Evaluator sends c_{input} and c_{output} to $\mathcal{G}_{\mathsf{trust}}$ for authentication.
$\mathcal{G}_{\mathsf{trust}}$ confirms receipt validity and hash storage; evaluator aborts if invalid.
for each input wire $i \in I$ corresponding to bit $x[i]$ **do**
    Prover provides garbled input g_i for w_i in GC_j ($j \notin J$).
    Evaluator sends g_i to $\mathcal{G}_{\mathsf{trust}}$.
    $\mathcal{G}_{\mathsf{trust}}$ retrieves h_{i,z_i} where $z_i = x[i]$.
    if $\mathsf{Hash}(g_i) = h_{i,z_i}$ **then** $\mathcal{G}_{\mathsf{trust}}$ accepts g_i and notifies evaluator.
    else $\mathcal{G}_{\mathsf{trust}}$ rejects g_i, notifies evaluator, and evaluator aborts.
    end if
end for
Evaluator computes garbled output Y using $\mathsf{Ev}(GC_j, g_i i \in I)$ for $j \notin J$.
Evaluator decodes Y to cleartext y using De and hashes garbled output components.
for each output wire $o \in O$ **do**
    Evaluator sends hashed garbled output for w_o to $\mathcal{G}_{\mathsf{trust}}$.
    $\mathcal{G}_{\mathsf{trust}}$ checks against $h_{o,\sigma}$ corresponding to decoded y.
    if mismatch occurs **then** $\mathcal{G}_{\mathsf{trust}}$ notifies evaluator, and evaluator aborts.
    end if
end for
Evaluator checks output consistency across GC_j for $j \notin J$; accepts if a majority agree on y.
return Cleartext output y if all validations and consistency checks pass.

Concretely, when using $\mathcal{G}_{\mathsf{trust}}$, we instantiate keys as tuples $\mathsf{key} = (\mathsf{sid}, \mathsf{pid}, C, i, \sigma)$, where sid is the UC session identifier, pid is the prover's identity, C is the circuit identifier, i is the wire index and $\sigma \in \{0, 1\}$ is the label bit. The commitment receipts c_{input} and c_{output} produced in Algorithms 1 and 2 are simply the collection of such keys together with their associated hashes stored via $\mathcal{G}_{\mathsf{trust}}.\mathsf{Put}$. Thus $\mathcal{G}_{\mathsf{trust}}$ itself does not need a special receipt type beyond first-write-wins storage.

2.6 Step-by-step overview of OHMG

Both parties commit to a circuit C computing F. We fix an elliptic curve $E(\mathbb{Z}_p)$ with prime p (specified in Section 4) and describe algorithms $(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev}) = \mathsf{OHMG}$.

Definition 6. Let G be a gate of circuit C; by combining Boolean gates, we consider *arithmetic* gates computing $f : \times_i \mathbb{Z}_{n_i} \to \times_j \mathbb{Z}_{m_j}$. The circuit's *wires* are primarily Boolean (for binary operations), but arrays of wires can carry arithmetic information (Definition 14). Wires carry data as yes/no values, with some groups representing numbers via binary codes or single active wires for complex calculations. A *garbling* of a Boolean wire w_i uses two correlated nonces: $w_i^0 = w_i(0) = R_i$ and $w_i^1 = w_i(1) = R_i\Delta \in \mathbb{Z}_p^*$, tied to bits 0 (off) and 1 (on). Wire array types include: *one-hot tensor products*, *one-hot matrices*, *one-hot vectors*, and *binary vectors*. For a gate computing $f : \mathbb{Z}_n \to \mathbb{Z}_m$, one-hot vectors with n input wires and m output wires (one active per vector) are useful, contrasting with binary vectors where all 1 wires are on. A *word* wire carries arithmetic information as $\Omega(a) = r\Delta^a \in \mathbb{Z}_p^*$; for instance, an output wire can encode integer a modulo m. A garbling of cleartext a is either a word wire or an array with selected active wires. Input wires are Boolean, grouped to encode arithmetic inputs binary. The $\mathsf{bin2hot}$ primitive converts garbled inputs to one-hot vectors for arithmetic gates. Instead of garbling gate truth tables,

we garble wires consistently, guiding the evaluator to compute garbled outputs $\breve{f}$. Some gates need no extra data, but others ($\mathcal{G}_k$) require a ciphertext help H_k for the evaluator to compute $\breve{f}_k$. For multi-variable functions, we construct a tensor product of one-hot input vectors (Definition 18), using H_k to evaluate it, enabling consistent computation. Transmitted data, the *material* of GC, includes generators and helps. The evaluator knows cleartexts a_i after garbler commitment, crucial for computing $\breve{f}_k$.

We describe the algorithms defining OHMG:

1. Algorithm En: The *encoding function* f_e, handled only by the garbler, uses input wire nonces R_i and Δ to set wires on/off per the binary cleartext inputs.En is presented in Algorithm 3 below.

Algorithm 3 Encoding Algorithm En

Require: Encoding information f_e from Gb, containing the global correlation Δ and the input-wire nonces $\{R_i\}_{i\in I}$. Cleartext input $x = (x_i)_{i\in I} \in \{0,1\}^{|I|}$.
Ensure: Garbled input $X = (w_i(x_i))_{i\in I}$.
1: **for** each input wire $i \in I$ **do** Compute $w_i(x_i) := R_i\Delta^{x_i} \in \mathbb{Z}_p^*$.
2: **end for**
3: **return** $X := (w_i(x_i))_{i\in I}$.

2. Algorithm De: The decoding function f_d operates at inputs or outputs: the evaluator hashes garbled inputs and compares them with $\mathsf{Hash}(w_i(\sigma))$ for $\sigma \in \{0,1\}$ and $i \in I$ to decode binary inputs; or she hashes garbled outputs and compares them with $\mathsf{Hash}(w_o(\sigma))$ for $o \in O$ to decode binary outputs. The committed hashes are retrieved from $\mathcal{G}_{\mathsf{trust}}$ where the garbler posted them. De is presented in Algorithm 4 below.

Algorithm 4 Decoding Algorithm De

Require: Decoding information f_d identifying the set of wires to decode (inputs I or outputs O). Garbled values $Y = (w_o)_{o\in O}$ (or $(w_i)_{i\in I}$).
Ensure: Cleartext bit-string $y = (y_o)_{o\in O} \in \{0,1\}^{|O|}$ or $\perp$.
1: **for** each wire $o \in O$ **do**
2: Retrieve $h_0 := \mathcal{G}_{\mathsf{trust}}.\mathsf{Get}(\mathsf{sid}, \mathsf{out}{:}o{:}0)$ and $h_1 := \mathcal{G}_{\mathsf{trust}}.\mathsf{Get}(\mathsf{sid}, \mathsf{out}{:}o{:}1)$.
3: Compute $h := \mathsf{Hash}(w_o)$.
4: **if** $h = h_0$ **then** Set $y_o := 0$.
5: **else if** $h = h_1$ **then** Set $y_o := 1$.
6: **else return** $\perp$ ▷ Decoding failure.
7: **end if**
8: **end for**
9: **return** $y := (y_o)_{o\in O}$.

3. Algorithm Ev: The garbler provides garbled inputs, and the evaluator decodes cleartexts using De. Ev is presented in Algorithm 5 below.

Algorithm 5 Evaluation Algorithm Ev

1: Compute all gates off-chain using cleartext inputs and garblings (privately on Valery's system).
2: Use helps H_k to compute gates with tensor products $T_\otimes$ of input wires for gate $\mathcal{G}_k$.
3: Use generators G_{i,j_i} for $\mathsf{bin2hot}$ functionality.
4: Use generators G'_i for $\mathsf{word2hot}$ functionality.
5: Apply De on output wire garblings to obtain cleartext output.

4. Algorithm Gb: It produces the garbled circuit material. It samples the global correlation Δ and the auxiliary scalars (K, the per-gate scalars k_i, and the per-arity scalars $k^{(I)}$), derives the family of generators $(G_c, G_{i,j_i}, \widetilde{G}^{(I)}_{-j}, G'_i)$ used by the gates, samples the input-wire nonces R_i, computes wire labels $w_i(z) = R_i\Delta^z$, and propagates wire labels through the circuit, computing per-gate helps H_k as specified for each gate type. Output-wire hashes $\mathsf{Hash}(w_o(0))$ and $\mathsf{Hash}(w_o(1))$ are posted to $\mathcal{G}_{\mathsf{trust}}$ so that the evaluator can later validate the decoded labels. Gb is presented in Algorithm 6 below.

Algorithm 6 Garbling Algorithm Gb

1: Sample $\Delta \xleftarrow{\$} \mathbb{Z}_q^*$. Compute the `word2hot` generators: $G' := [K]G$ and $G'_i := [\Delta^i]G'$ for $i \in \{\pm 1, \ldots, \pm(n-1)\}$ (with $K \xleftarrow{\$} \mathbb{Z}_q^*$; here n denotes the largest `word2hot` length occurring in C, e.g. $n = 4m - 5$ when π_{mult} is used, and the same convention applies to the K-family of Definition 42). No unblinded power $[\Delta^c]G$ with $c \neq 0$ is ever computed or transmitted; all Δ-powers handed to the evaluator are blinded by a per-gate or per-arity secret (next step).
2: **Per-gate generators.** For each binary gate of bitwidth d_i ($i \in \{1, \ldots, l\}$): sample $k_i \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $G_{i,j_i} := [\Delta^{j_i} k_i]G$ for $j_i \in \{-(d_i - 1), \ldots, 0\}$; the power $G_{i,-d_i} = [\Delta^{-d_i} k_i]G$ is withheld. For each tensor-product arity $I \in \{2, \ldots, I_{\max}\}$ in C: sample $k^{(I)} \xleftarrow{\$} \mathbb{Z}_q^*$, set $\widetilde{G}^{(I)} := [k^{(I)}]G$ and $\widetilde{G}^{(I)}_{-j} := [\Delta^{-j}]\widetilde{G}^{(I)}$ for $j \in \{1, \ldots, I-1\}$.
3: **Wire labels.** Group C's input wires as N arrays of length d. Sample input nonces $R_i \in \mathbb{Z}_p^*$, set $w_i(z) := R_i\Delta^z$ for $z \in \{0,1\}$, and compute $\mathsf{Hash}(w_i(0)), \mathsf{Hash}(w_i(1))$ for input and output wires. Propagate nonces and compute per-gate helps H_k.
4: Post output-wire hashes via $\mathcal{G}_{\mathsf{trust}}.\mathsf{Put}(\mathsf{sid}, \mathsf{out}{:}o{:}\sigma, \mathsf{Hash}(w_o(\sigma)))$ for $o \in O, \sigma \in \{0,1\}$. The evaluator retrieves them via $\mathcal{G}_{\mathsf{trust}}.\mathsf{Get}$ to validate the decoded labels.
5: Send garbled material $\big(\{G_{i,j_i}\}, \{\widetilde{G}^{(I)}_{-j}\}, \{G'_i\}, \{H_k\}, \text{hashes}\big)$ to the evaluator.

5. Algorithm ev: the *plaintext evaluation algorithm*, the unmasked counterpart of Ev: gate by gate in topological order, it evaluates C on the cleartext input x and returns $y = F(x)$. It is never invoked by the protocol; it is the reference computation for correctness (Lemma 1) and authenticity (Definition 4).

Lemma 1. *Fix a security parameter κ and assume $\mathsf{Hash} : \mathbb{Z}_p \to \{0,1\}^\lambda$ is collision-resistant and preimage-resistant (Definition 40); Hash is not modelled as a random oracle in this paper. Let C be a circuit computing a function F over the gate types supported by OHMG. Then for every input x to C, an honest run of OHMG satisfies $\Pr\Big[\mathsf{De}\big(f_d, \mathsf{Ev}(GC, \mathsf{En}(f_e, x))\big) = \mathsf{ev}(F, x)\Big] \geq 1 - \mathsf{negl}(\kappa)$, where the probability is over the garbler's random choices $\Delta, \{R_i\}, \{k_i\}, K, \{k^{(I)}\}$ in Gb (Algorithm 6). The negligible term $\varepsilon_{\mathsf{coll}}(\kappa)$ accounts solely for the probability of an unintended collision in Hash during decoding; conditional on no such collision, equality holds identically.*

Proof. Fix the random choices of Gb; all algorithms are then deterministic. By induction on a topological order of C, every wire produced by Ev carries the label of its cleartext value: the base case is Algorithm 3, and the inductive step is the consistency theorem of the corresponding gate type (Theorem 1 for single-input gates, Theorems 2 and 3 for tensor-product gates, and the constructions of Section 4 for higher arities and the conversion gadgets). At the outputs, De compares $\mathsf{Hash}(w_o(y_o))$ against the two committed hashes; since $w_o(0) \neq w_o(1)$, decoding can fail only on an explicit collision of Hash, an event of probability at most $\varepsilon_{\mathsf{cr}}(\kappa) =: \varepsilon_{\mathsf{coll}}(\kappa)$, exhibited by the efficient procedure that runs Gb and compares the two output-wire hashes. Conditional on no collision, equality holds identically. □

3 One-hot encodings and cleartexts

This section is an overview of some ideas behind Heath's papers [Hea24] for operations in $\mathbb{Z}_p$ on one-hot encodings, and their approach to garbled circuits. All statements in this section are about cleartexts, however in the approach taken by Heath and Kolesnikov, most of it can be used for ciphertexts, masking the cleartexts x with a uniform mask α. This is possible in their scheme, given certain homomorphic features of their garblings. This masking, as far as we know, is not possible in our OHMG scheme presented in Section 4, which is designed for privacy-free applications. We begin this section with some notations:

We will be dealing with numbers given in its arithmetic expression (that is a word $a \in \mathbb{Z}_n$) but we will also be interested in their binary expression $\mathsf{bin}(a) \in \{0,1\}^d$. Therefore it will be convenient to distinguish two one-hot encodings, first the one for the binary expression (of length d) and then the one for the arithmetic expression (of length $n = 2^d$).

Definition 7. Let $\mathcal{H} : \{0,1\}^d \to \{0,1\}^{2^d}$ be the map that assigns to a binary expression $z \in \{0,1\}^d$ the vector with only 1 in the position corresponding to z, and every other entry is 0. For $d = 1$ we map $0 \mapsto (1,0), 1 \mapsto (0,1)$, while for $d = 2$ we map $00 \mapsto (1,0,0,0),\ 01 \mapsto (0,1,0,0),\ 10 \mapsto (0,0,1,0),\ 11 \mapsto (0,0,0,1)$.

It will be convenient to count indices from $i = 0$; that is an element $x \in \{0,1\}^k$ will be denoted $x = (x_0, x_1, \ldots, x_{k-1})$, hence the "first place" is in fact x_0. We will denote $\mathcal{H}(z)_k \in \{0,1\}$, $z \in \{0,1\}^d$, $k \in \mathbb{Z}_{2^d}$ to the k-th entry of $\mathcal{H}(z)$.

Definition 8. For a Boolean function $f : \{0,1\}^d \to \{0,1\}^n$ we denote by $f(z)_j$ to the j-th entry of $f(z)$, $z \in \{0,1\}^d$. We also denote with $\mathcal{T}(f)$ the $2^d \times n$ matrix (2^d rows and n columns) given by its truth table, with inputs and outputs ordered increasingly, so the i-th row is the full bit string $f(z)$.

Remark 6. If $a \in \{0,1\}^n$ and $f : \{0,1\}^n \to \{0,1\}^d$, then $\mathcal{T}(f)^T \cdot \mathcal{H}(a)^T = f(a)^T$. Indeed, if we multiply $\mathcal{T}(f)^T$ on the right by the vector column $\mathcal{H}^T$ (which is only non-zero in the a-th position), we recover its a-th column, which is exactly $f(a)$ written as a column.

Remark 7. For $f : \{0,1\}^n \to \{0,1\}^d$ consider the linear mapping $\widehat{f} : \mathbb{Z}_2^{2^n} \to \mathbb{Z}_2^d$ defined on the basis $\{\mathcal{H}(a)\}_{a \in \{0,1\}^n}$ as $\widehat{f}(\mathcal{H}(a)) = f(a)$ (Remark 6). Then $\mathcal{T}(f)^T$ is just the matrix of $\widehat{f}$ in the canonical bases that is $M_{\widehat{f}} = \mathcal{T}(f)^T$. Therefore Remark 6 can be interpreted as the standard computation of $\widehat{f}$ using its matrix.

Definition 9. Also known as the *outer product* of vectors, for a given ring $\mathbb{F}$ and $a \in \mathbb{F}^n$, $b \in \mathbb{F}^m$, it is defined as the matrix $a \otimes b = a^T \cdot b = (a_i b_j)_{ij} \in \mathbb{F}^{n \times m}$. Note that there is an identification of $a \otimes b$ with a bilinear mapping from $\mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}$, implemented by $(x, y) \mapsto x \cdot a^T \cdot b \cdot y^T = x \cdot a^T \cdot y \cdot b^T = a \cdot x^T \cdot b \cdot y^T$. The coefficients $a_i b_j$ can be recovered evaluating this bilinear mapping in the canonical bases.

Definition 10. In general, if $a_i \in \mathbb{F}^{n_i}$ for $i \in \{0, \ldots, I-1\}$, the entries of their *tensor product* are indexed by the number of inputs as follows:

$$(a_0 \otimes \cdots \otimes a_{I-1})_{k_0 k_1 \cdots k_{I-1}} = \prod_{i=0}^{I-1} (a_i)_{k_i}$$

where $k_i \in \mathbb{Z}_{n_i}$ and $(x)_k$ indicates the k-th entry of x.

As before, we can identify any tensor product $a_1 \otimes \cdots \otimes a_I$ with a multi-linear mapping

$$(x_1, \ldots, x_I) \mapsto x_1 \cdot a_1^T x_2 \cdot a_2^T \ldots x_I \cdot a_I^T \in \mathbb{F}, \qquad x_i \in \mathbb{F}^{n_i}.$$

Lemma 2. *Let $x \in \mathbb{F}^k, y \in \mathbb{F}^r$, let $g : \mathbb{F}^k \to \mathbb{F}^d$ and $h : \mathbb{F}^r \to \mathbb{F}^s$ be linear. Let M_g, M_h be their respective matrices in the canonical bases. Then $M_g(x \otimes y) = g(x) \otimes y$ and $(x \otimes y)M_h^T = x \otimes h(y)$.*

Proof. Immediate from the associativity of the matrix product: $M_g \cdot (x^T \cdot y) = (M_g \cdot x^T) \cdot y = g(x) \otimes y$, and symmetrically for M_h^T. □

Corollary 1. *Let $a \in \{0,1\}^n$, $b \in \{0,1\}^m$. If $f : \{0,1\}^n \to \{0,1\}^d$ then*

$$\mathcal{T}(f)^T(\mathcal{H}(a) \otimes b) = f(a) \otimes b$$

while if $f : \{0,1\}^m \to \{0,1\}^k$ then $(a \otimes \mathcal{H}(b))\mathcal{T}(f) = a \otimes f(b)$.

In particular if $f = id_n$ is the identity map of $\{0,1\}^n$ (resp. of $f = id_m$ of $\{0,1\}^m$), then

$$\mathcal{T}(id_n)^T \cdot (\mathcal{H}(a) \otimes b) = a \otimes b = (a \otimes \mathcal{H}(b))\mathcal{T}(id_m).$$

Proof. Apply the previous lemma with $g = \widehat{f}$ (Remark 7), $x = \mathcal{H}(a)$, $y = b$ for the first assertion, and symmetrically with $h = \widehat{f}$ for the second; the identity-map cases follow at once. □

Definition 11. The *arithmetic one-hot encoding* $\delta : \mathbb{Z}_n \to \mathbb{Z}_2^n$ is given by

$$a \mapsto \delta(a) = (\delta_0(a), \ldots, \delta_{n-1}(a)) = (a_0, \ldots, a_{n-1}) \in \{0,1\}^n,$$

where $\delta_i(a) = \delta_{i,a}$ that is $a_i = 1$ in position a, and otherwise $a_i = 0$.

Remark 8. Note that for $n = 2$ both one-hot encodings agree, that is $\mathcal{H} = \delta : \{0,1\} = \mathbb{Z}_2 \to \{0,1\}^2 = \mathbb{Z}_2^2$. Moreover, when $n = 2^d$, then from the very definitions we have

$$\delta(a) = \mathcal{H}(\mathsf{bin}(a)) \qquad a \in \mathbb{Z}_n. \tag{1}$$

In the general case ($n < 2^d$), $\delta(a)$ is the truncation of $\mathcal{H}(\mathsf{bin}(a))$ to its first n coordinates.

Remark 9. If $f : \mathbb{Z}_n \to \{0,1\}^d$, then we say that

$$\mathcal{T}(f) = \begin{pmatrix} f(0)_0 & f(0)_1 & \cdots & f(0)_{d-1} \\ f(1)_0 & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ f(n-1)_0 & \cdots & \cdots & f(n-1)_{d-1} \end{pmatrix}$$

is the *truth table* of f. Reasoning as in binary one-hot (with linear $\hat{f}$ defined on the basis as $\hat{f}(\delta(a)) = f(a)$, $a \in \mathbb{Z}_n$), for each $a \in \mathbb{Z}_n$ we have

$$\mathcal{T}(f)^T \cdot \delta(a)^T = f(a)^T. \tag{2}$$

In particular for $f = \mathsf{bin} : \mathbb{Z}_n \to \{0,1\}^d$ (with $d = \lceil \log_2 n \rceil$) the map assigning to each $a \in \mathbb{Z}_n$ its binary expression, we have $\mathcal{T}(\mathsf{bin})^T \delta(a)^T = \mathsf{bin}(a)^T$.

Mixing binary encodings with one-hot encodings, we now have an analogue of Corollary 1, replacing the binary one-hot encoding $\mathcal{H}$ with the arithmetic one-hot encoding δ, with an identical proof:

Corollary 2. *If $f : \mathbb{Z}_n \to \{0,1\}^d$ then $\mathcal{T}(f)^T(\delta(a) \otimes b) = f(a) \otimes b$, for $a \in \mathbb{Z}_n$, $b \in \mathbb{Z}_s^l$. If $f : \mathbb{Z}_m \to \{0,1\}^k$, then $(a \otimes \delta(b))\mathcal{T}(f) = a \otimes f(b)$ for $a \in \mathbb{Z}_r^l$, $b \in \mathbb{Z}_m$. In particular if $f = \mathsf{bin} : \mathbb{Z}_* \to \{0,1\}^*$, then $\mathcal{T}(\mathsf{bin})^T(\delta(a) \otimes b) = \mathsf{bin}(a) \otimes b$ and*

$$\mathcal{T}(\mathsf{bin})^T(\delta(a) \otimes \delta(b))\mathcal{T}(\mathsf{bin}) = \mathsf{bin}(a) \otimes \mathsf{bin}(b).$$

3.1 Computing functionalities in one-hot encoding

In this section we are interested in discussing the computation of a function f when the input is encoded in one-hot, and we *also want a one-hot output* (compare with Remarks 6 and 9). We begin with the simplest case, whose proof is straightforward:

Lemma 3. *If we have $f : \mathbb{Z}_n \to \mathbb{Z}_m$, then $\delta_j(f(a)) = \sum_{i \in \mathbb{Z}_n \,:\, f(i)=j} \delta_i(a)$ for $j \in \mathbb{Z}_m$. If $f : \mathbb{Z}_n \times \mathbb{Z}_m \to \mathbb{Z}_r$, then $\delta_k(f(a,b)) = \sum_{i,j \,:\, f(i,j)=k} \delta_i(a)\delta_j(b) = \sum_{i,j \,:\, f(i,j)=k} (\delta(a) \otimes \delta(b))_{ij}$ for $k \in \mathbb{Z}_r$.*

Remark 10.

1. These ideas extend componentwise and multilinearly to $f : \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_I} \to \mathbb{Z}_{k_1} \times \cdots \times \mathbb{Z}_{k_J}$; we pursue only the one case used later, a constructive translation from binary encodings to arithmetic one-hot encodings.

2. For $h_i = ((h_i)_0, (h_i)_1) \in \{0,1\}^2$, consider the tensor product

$$(\bigotimes_{i \in \mathbb{Z}_d} h_i)_X = (h_0 \otimes \cdots \otimes h_{d-1})_{x_0 \ldots x_{d-1}} = \prod_{i=0}^{d-1} (h_i)_{x_i} \qquad X = x_0 \ldots x_{d-1} \in \{0,1\}^d.$$

   Note that for each $h = (h_0, \ldots, h_{d-1})$ there is exactly one X which makes the product equal to 1, and otherwise the product is zero.

Lemma 4. *Let $f : \{0,1\}^d \to \{0,1\}^n$, $Z = z_0 \ldots z_{d-1} \in \{0,1\}^d$. Then for $\mathcal{H} : \{0,1\} \to \{0,1\}^2$ and each $k \in \mathbb{Z}_n$ we have*

$$f(Z)_k = \sum_{X \in \{0,1\}^d \,:\, f(X)_k = 1} (\mathcal{H}(z_0) \otimes \cdots \otimes \mathcal{H}(z_{d-1}))_X.$$

In particular if $n = 2^d$ then for $a \in \mathbb{Z}_n$ and each $k \in \mathbb{Z}_n$ we have

$$\delta_k(a) = (\mathcal{H}(a[0]) \otimes \cdots \otimes \mathcal{H}(a[d-1]))_{\mathsf{bin}(k)}$$

where $a[0] \ldots a[d-1] = \mathsf{bin}(a)$ as usual.

Proof. The tensor entry $(\bigotimes_i \mathcal{H}(z_i))_X$ equals 1 iff $X = Z$ and 0 otherwise, so both sides agree by inspection. The second assertion is the case $f = \mathcal{H}$, using $\delta_k(a) = (\mathcal{H} \circ \mathsf{bin})(a)_k$ and that $\mathcal{H}(X)_k = 1$ exactly for $X = \mathsf{bin}(k)$. □

3.2 Dictionaries: cleartexts

In this section we describe the (well-known) implementations that give translations between four expressions of a number $a \in \mathbb{Z}$: the binary expression, the (arithmetic) one-hot expression, the (binary) one-hot expression and the *arithmetic* expression (this is the standard decimal expression of a, also called *word* in [Hea24]).

1. *Arithmetic to binary, binary to arithmetic, and arithmetic one-hot to arithmetic*: implemented by the usual maps $a \mapsto \mathsf{bin}(a)$, $\mathsf{bin}(a) \mapsto \sum_i a[i]2^i$, and $\delta(a) \mapsto \sum_i i \cdot a_i$ respectively.

2. *Arithmetic one-hot to binary*: Let $\mathsf{bin} : \mathbb{Z}_n \to \{0,1\}^d$ (with $d = \lceil \log_2 n \rceil$) be the map assigning to each $a \in \mathbb{Z}_n$ its binary expression, let $\mathcal{T}(\mathsf{bin})$ be its truth table (Remark 9). Then translation from one-hot to binary is given by

$$(a_0, \cdots, a_{n-1}) = \delta(a) \mapsto \mathcal{T}(\mathsf{bin})^T \delta(a)^T = \mathsf{bin}(a)^T.$$

Definition 12. For $X \in \{0,1\}^d$ we introduce the operator $T_X : \{0,1\}^d \to \{0,1\}^d$ defined as $(T_X Y)[i] = Y[i]X[i] + \overline{Y[i]}\ \overline{X[i]}$ for $Y \in \{0,1\}^d$. Note that T_X flips bits of Y: the i-th bit of Y is flipped if and only if the i-th bit of X is zero (otherwise it is left untouched). Fix n and for $d = \lceil \log_2(n) \rceil$ we let $\mathsf{b2h} : \{0,1\}^d \to \{0,1\}^n$ be given by $\mathsf{b2h} : Y = \mathsf{bin}(a) \mapsto \prod_{i=0}^{d-1} (T_{\mathsf{bin}(k)} Y)[i] = \delta_k(a)$ for $k \in \mathbb{Z}_n$.

Remark 11. This alternative presentation using tensor products will be useful later: we leave to the reader to check that

$$\prod_{i=0}^{d-1} (T_X Z)[i] = (\mathcal{H}(z_0) \otimes \cdots \otimes \mathcal{H}(z_{d-1}))_X \qquad \text{for any } X, Z \in \{0,1\}^d.$$

Then by means of Lemma 4 and the previous equality it is easy to check that

Lemma 5. *The map* $\mathsf{b2h} : Z \mapsto (\mathcal{H}(z_0) \otimes \cdots \otimes \mathcal{H}(z_{d-1}))_{\mathsf{bin}(k)}$ *for* $k \in \mathbb{Z}_n$ *implements a translation from the binary expression* $Z = \mathsf{bin}(a)$ *of* $a \in \mathbb{Z}_n$ *to the arithmetic one-hot encoding of* a*, that is:* $\mathsf{b2h}(\mathsf{bin}(a)) = \delta(a), \mathsf{bin}(a) \in \{0,1\}^d, d = \lceil \log_2 n \rceil$.

4 OHMG: One-hot modular garbling

In this section we describe how to represent numbers and functions in finite fields, and we describe our proposed mechanism for describing a garbled tensor product that will allows to garble *any function* at each gate G_i with message cost $O(1)$. Our garbling scheme OHMG is based on the tensor product of arithmetic one-hot garblings.

Remark 12. As we described in the introduction, the evaluator knows the garblings of the input wires, but also their cleartexts. The inputs are chosen by the garbler, who also garbles the circuit. Then the garbler provides to the evaluator a ZKP that this garbled circuit if actually computing the chosen (agreed upon both parties) circuit C.

Remark 13. Throughout this paper we will be using the elliptic curve secp256k1, given by the Weierstrass equation $y^2 = x^3 + 7$ over the finite field $\mathbb{Z}_p$, with $p = 2^{256} - 2^{32} - 977$. We let G be a generator in the curve, and we use multiplicative notation for the curve operation that is $[2]G = G \cdot G$. For the garbling of all the wires of a full circuit, we fix a *correlation* $\Delta \in \mathbb{Z}_p^*$.

Definition 13. A value $a \in \mathbb{Z}_p^*$ is *EC-safe* if it satisfies $a \neq 0 \pmod q$, where q is the order of the generator G of the secp256k1 curve. This ensures $[a]G$ is not mapped to the origin of the group. We will ask that all our labels for the wires are EC-safe. For the global correlation Δ, additionally we require $\Delta \neq 1 \pmod q$, ensuring $[\Delta]G \neq G$ is not a trivial hidden correlation.

Remark 14. In applications, since the order of the curve is quite similar to p, we have that for $a \in \mathbb{Z}_p^*$, we have that $a \pmod q$ maps to $\{0, \ldots, q-1\}$ with bias $O((p-q)/p) \approx 2^{-128}$. Hence $a = 0 \pmod q$ with probability $\approx 2^{-256}$, which is negligible. A similar reasoning can be applied to the condition $\Delta \neq 1 \pmod q$, so from now we will simply say $R, \Delta \in \mathbb{Z}_p^*$. So all non-zero randomly chosen correlation and nonces are statistically EC-secure.

In protocols requiring field inverses in $\mathbb{Z}_p^*$ (for instance, Δ^{-1} for $G_{-1} = [\Delta^{-1}]G$ in tensor products and not), the effective scalar multiplication is mod q. For $a \in \mathbb{Z}_p^*$, $a^{-1} = b$ mod p satisfies $a \cdot b = 1 + kp$. Then $[(a \cdot b)]G = G + k \cdot ([p]G) = G + [k \cdot r]G$, where $r = p$ mod $q \neq 0$ for secp256k1. This equals G if, and only if, $kr \equiv 0 \pmod q$, that is, $k \equiv 0$ mod q (since r is invertible mod q). For random a, the probability that $k \equiv 0 \pmod q$ is negligible. Thus, mod-p field inverses would *fail* to act as group inverses with overwhelming probability: this is precisely why all wire labels, masks, products and inverses in this paper are computed in $\mathbb{Z}_q^*$ (Section 2.1), where scalar arithmetic and group arithmetic agree by construction. No protocol computation is performed modulo p.

4.1 Wires, nonces, and vectors

Definition 14. Our garbling scheme makes use of two types of garblings:

1. Boolean wire: conformed by a random nonce $R \in \mathbb{Z}_p^*$ and boolean variable $z \in \{0,1\}$, defined as $w(z) = R\Delta^z$.
2. Word wire: conformed by a random nonce $r \in \mathbb{Z}_p^*$ and a modular variable $x \in \mathbb{Z}_n$, defined as $\Omega(x) = r\Delta^x$.

We say that the Boolean wire is *on* (or *hot*) when $z = 1$ and its *off* (or *cold*) when $z = 0$. Let $\mathcal{W}$ be the set of all the binary wires of the circuit C, numbered in the set $\{0, 1, \ldots, W-1\}$. For $k \in \mathcal{W}$, we denote $w_k(z) = R_k\Delta^z$ the k-th wire of the circuit.

Remark 15. As we already mentioned, we will not be using the standard notation for a Boolean garbled wire turned off $w_i^0 = R_i$ or turned on $w_i^1 = R_i\Delta$, since the super-index will collide with the exponentiation of numbers in $\mathbb{Z}_p$.

For the following definition, recall that $\delta_i(x) = 0$ for $i \neq x$ and $\delta_x(x) = 1$ that is δ is Dirac's delta function.

Definition 15. Wires can be arranged in certain specific ways, for convenience:

1. A *binary vector of wires* with d-digits is an ordered set of d Boolean wires, denoted $B(Z) = (w_0(z_0), w_1(z_1), \cdots, w_{d-1}(z_{d-1}))$, for $Z = z_0 z_1 \cdots z_{d-1} \in \{0,1\}^d$. We denote with $B(Z)_i = w_i(z_i) = R_i\Delta^{z_i}$ to the i-th wire of the vector.
2. A *one-hot vector of wires* is the composition of the arithmetic one-hot encoding map $\delta : \mathbb{Z}_n \to \mathbb{Z}_2^n$ with a binary n-vector of wires:

$$\begin{aligned} W(x) = (B \circ \delta)(x) &= (w_0(\delta_0(x)), \ldots, w_{n-1}(\delta_{n-1}(x)) \\ &= (R_0\Delta^{\delta_0(x)}, \ldots, R_{n-1}\Delta^{\delta_{n-1}(x)}), \qquad x \in \mathbb{Z}_n. \end{aligned}$$

   Equivalently, it is a vector of wires, with one and only one-hot entry. We denote $W(x)_j = R_j\Delta^{\delta_j(x)}$ to the j-th wire of the vector.

Definition 16. If $A = \{i_0, \ldots, i_{d-1}\}$ is an *ordered* subset of the index set $\mathcal{W}$ of the binary wires, we indicate with $B_A = (w_{i_0}, w_{i_1}, \ldots, w_{i_{d-1}})$, and $B_A(Z)_k = w_{i_k}(z_{i_k}) = R_{i_k}\Delta^{z_{i_k}}$. We let $W_A = (w_{i_0} \circ \delta_0\,,\, w_{i_1} \circ \delta_1\,,\, \ldots\,,\, w_{i_{d-1}} \circ \delta_{d-1})$ and write $W_A(x)_k = R_{i_k}\Delta^{\delta_k(x)}$, where it is assumed that $x \in \mathbb{Z}_d$.

For $k = 0, \ldots, d-1$, we denote $w_{A,k} := w_{i_k}, R_{A,k} := R_{i_k}, i_k \in A$ when possible, to make explicit the set A, and avoid the double sub-index notation. With this notation $W_A(x)_k = R_{i_k}\Delta^{\delta_k(x)} = w_{A,k}(x) = R_{A,k}\Delta^{\delta_k(x)}$. Note the two possible notations $W_A(x)_k = w_{A,k}(x)$, each one will be used in the most convenient scenario (that is when we want to emphasize that we are picking a coordinate of the vector, or when we are making computations with coordinates.

Remark 16. For the input wires, which are all Boolean, the garbler chooses randomly all of their nonces. The nonces for all the other wires are obtained from these ones, by going through a procedure for each gate G_i, in topological order of the gates. The procedure for each type of gate will be specified below. The following auxiliary notions will also be useful.

Definition 17. Let $a \in \mathbb{Z}_n$

1. A *word garbling* of cleartext $a \in \mathbb{Z}_n$ is obtained by evaluating a word wire $\Omega(a) = r\Delta^a$.
2. A *binary garbling* of cleartext $a \in \mathbb{Z}_n$ is obtained by evaluating a binary vector of wires at $Z = \mathsf{bin}(a) = a[0] \ldots a[d-1]$, that is $B(\mathsf{bin}(a)) = (R_0\Delta^{a[0]}, \ldots, R_{d-1}\Delta^{a[d-1]})$.

3. A *one-hot garbling* of a is obtained by evaluating a one-hot vector of wires, that is $W(a) = (R_0\Delta^{\delta_0(a)}, \ldots, R_{n-1}\Delta^{\delta_{n-1}(a)})$.

Definition 18. A one-hot *tensor product* of order 2 is a $n \times m$ matrix (also called *one-hot matrix*) of Boolean wires $w_{i,j}(z_{i,j}) = R_{i,j}\Delta^{z_{i,j}}, z_{i,j} \in \{0,1\}, i \in \mathbb{Z}_n, j \in \mathbb{Z}_m$ with $R_{i,j} \in \mathbb{Z}_p^*$ and all of its entries but one turned off (that is it has only one-hot entry).

Given cleartexts $a \in \mathbb{Z}_n$, $b \in \mathbb{Z}_m$, a *one-hot garbled tensor product* is obtained by setting the hot entry of T exactly at position $i = a$, $j = b$; this ordered set of garblings is denoted as $T(a,b)$, and its i,j entry is denoted as $T(a,b)_{ij}$.

In general, a *one-hot tensor product* of order d is an indexed collection of Boolean wires

$$T = R_{i_0,\ldots,i_{d-1}}\Delta^{z_{i_0,\ldots,i_{d-1}}}, \quad z_* \in \{0,1\}, \quad i_k \in \mathbb{Z}_{n_k}, \quad R_* \in \mathbb{Z}_p^*,$$

with all of its entries but one turned off (that is it has only one-hot entry).

Given cleartexts $a_k \in \mathbb{Z}_{n_k}$, we set the hot entry of T at $i_k = a_k$ for all $k = 0, \ldots, d-1$ to obtain a *one-hot garbled tensor product* of the cleartexts, denoted $T(a_0, \ldots, a_{d-1})$. it is $(i_0, \ldots, i_{d-1})$-th entry will be denoted as $T(a_0, \ldots, a_{d-1})_{i_0 \ldots i_{d-1}}$.

We now describe the available gates, how to compute the nonces of the output wires knowing the nonces of the input wires (done by the garbler), and how to compute the garbled output given the garbled input (done by the evaluator). We will do this for several types of functions.

Definition 19. For a given gate G of circuit C computing the functionality f, we will denote with $\breve{f}$ the operation on garblings that produced the garbled output of gate G.

4.2 Garbling $f : \mathbb{Z}_n \to \mathbb{Z}_m$ with one-hot inputs and outputs

Let G be the gate computing f, let W_I, W_O be its (one-hot) input and output vectors of wires.

Definition 20. Let $I, O \subset \mathcal{W}$ be sets of Boolean wires of circuit C, the first of cardinal n and the second of cardinal m. Given the nonces $R_{I,i}$ ($i \in \mathbb{Z}_n$) of the one-hot input vector of wires W_I, the nonces $R_{O,k}$ for the one-hot output vector of wires W_O are defined (computed by the garbler) as follows: $R_{O,k} = \prod_{i \in \mathbb{Z}_n \,:\, f(i)=k} R_{I,i}, \qquad k \in \mathbb{Z}_m$.

Definition 21. The garbled output of the gate G computing f with garbled input $W_I(a)$ is $\breve{f} : \mathbb{Z}_p^n \to \mathbb{Z}_p^m$ defined as

$$\breve{f}(W_I(a))_k := \prod_{i \in \mathbb{Z}_n \,:\, f(i)=k} W_I(a)_i = \prod_{i \in \mathbb{Z}_n \,:\, f(i)=k} R_{I,i}\Delta^{\delta_i(a)} \tag{3}$$

For the following theorem (and every time we prove consistency) it is convenient to have in mind Definition 3, where consistency its just a way we express correctness in our approach.

Theorem 1. *The garbled one-hot vector output of gate G computing $f(a)$ is the output vector of wires of gate G evaluated at $f(a)$. That is: $W_O(f(a)) = \breve{f}(W_I(a))$.*

Proof. Since the vector of wires W_O has nonces $R_{O,k}$: $W_O(f(a))_k = R_{O,k}\Delta^{\delta_k(f(a))}$ for each $k \in \mathbb{Z}_m$. That is $W_O(f(a))_k = \prod_{i \in \mathbb{Z}_n \,:\, f(i)=k} R_{I,i}\Delta^{\delta_k(f(a))}$.

But $\delta_k(f(a)) = 0$ unless $k = f(a)$, in which case $\delta_k(f(a)) = 1$, hence

$$W_O(f(a))_k = \begin{cases} \prod_{i \in \mathbb{Z}_n \,:\, f(i)=k} R_{I,i} = \prod_{i \in \mathbb{Z}_n \,:\, f(i)=k} R_{I,i}\Delta^{\delta_i(a)} & k \neq f(a) \\ \prod_{i \in \mathbb{Z}_n \,:\, f(i)=f(a)} R_{I,i}\Delta = \prod_{i \in \mathbb{Z}_n \,:\, f(i)=f(a)} R_{I,i}\Delta^{\delta_i(a)} & k = f(a) \end{cases};$$

the equality in the first line because in this case a does not belong to the product index set, hence $\delta_i(a) = 0$. The equality in the second line is because in this case, a belongs to the product index set, with one occurrence exactly when $i = a$. Comparing with (3), we have completed the proof of the theorem. □

Remark 17. As mentioned, this functionality can be implemented for free (no material is transmitted). It is a multiplicative analogue of (2) and Lemma 3. Note that at this point the evaluator, knowing a and f, knows the cleartext $f(a)$ hence she knows the garbling of this output that is she knows the exact *position* $b = f(a)$ which is hot in this garbled vector of wires. Hence she can use this as the input of another functionality.

Lemma 6. *Let $f : \mathbb{Z}_n \to \mathbb{Z}_m$ and $g : \mathbb{Z}_m \to \mathbb{Z}_d$. Then $\widetilde{g \circ f} = \breve{g} \circ \breve{f}$.*

Proof. Let W_I be the input vector of wires of the first gate computing f. First note that $\breve{g}(\breve{f}(W_I(a)))_\ell = \prod_{k\,:\,g(k)=\ell} \prod_{i\,:\,f(i)=k} W_I(a)_i$. On the other hand

$$\widetilde{(g \circ f)}(W_I(a))_\ell = \prod_{i\,:\,g(f(i))=\ell} W_I(a)_i.$$

The proof then follows from the elementary set equality:

$$\{i \in \mathbb{Z}_n : g(f(i)) = \ell\} = \bigsqcup_{k \in \mathbb{Z}_m} \{i \in \mathbb{Z}_n : f(i) = k \ \ \textbf{AND} \ \ g(k) = \ell\},$$

where $\sqcup$ indicates disjoint union of sets. □

4.3 Garbling $f : \mathbb{Z}_n \times \mathbb{Z}_m \to \mathbb{Z}_d$ with one-hot inputs and outputs

We move on to gates that have two one-hot vectors of wires as inputs W_A, W_B, and produce a one-hot vector of wires as output W_O. This is the most relevant case (conceptually, in terms of presentation and interpretation) of our scheme. The first step will be to define the output nonces, then we will need to define the auxiliary tensor product of the input wires.

Definition 22. In the security analysis, we model $X : \mathcal{G} \to \mathbb{Z}_q^*$ as a random oracle, realized by querying the global functionality $\mathcal{G}_{\mathsf{RO}}^X$ (Algorithm 8 in Section 5.2 below) on inputs $(\mathsf{sid}, \mathsf{tag}, U)$, where $U \in \mathcal{G}$ is a canonically encoded group element, sid is the session identifier, and tag is a domain-separation string specifying the protocol component and role. For $s \in \mathbb{Z}_q$ write $[s]G \in \mathcal{G}$ for scalar multiplication. All security theorems in Section 5 are stated in this $\mathcal{G}_{\mathsf{RO}}^X$-hybrid model.

Definition 23. For implementation, we instantiate X via one of the following constructions:

1. **Hash-to-field:** $X(U) := H(\mathsf{encode}(U)) \mod q$, where $H : \{0,1\}^* \to \{0,1\}^{2\kappa}$ is a cryptographic hash function (e.g., SHA-256 with output extension) and $\mathsf{encode} : \mathcal{G} \to \{0,1\}^*$ is a canonical injective encoding of group elements.
2. **Coordinate extraction:** $X(U) := x(U) \mod q$, where $x : \mathcal{G} \setminus \{\mathcal{O}\} \to \mathbb{Z}_p$ extracts the x-coordinate of the affine representation of $U \in E(\mathbb{Z}_p)$.

Remark 18. For elliptic curve operations, such as $X([r\Delta^x]G)$, we must ensure that $[\Delta^x]G \neq G$, as this could lead to trivial encodings or repeated generators, potentially compromising indistinguishability. For $[\Delta^x]G = G$, it must be $\Delta^x \equiv 1 \pmod q$. In the multiplicative group $\mathbb{Z}_q^*$ (of order $q-1$), this holds if, and only if, $\mathrm{ord}(\Delta) \mid x$. For a fixed exponent $x \neq 0$ and $\Delta \xleftarrow{\$} \mathbb{Z}_q^*$, the number of Δ with $\Delta^x = 1$ is $\gcd(x, q-1) \leq |x|$, hence $\Pr[\Delta^x \equiv 1 \bmod q] \leq |x|/(q-1)$. All exponents used by the protocol are polynomially bounded (wire cleartexts, $|x| \leq n$), so this probability is cryptographically negligible, ensuring that word wire encodings remain secure and distinct.

4.3.1 Per-arity generators for tensor products

For each arity $I \geq 2$ that occurs in the circuit, the garbler samples an independent secret scalar $k^{(I)} \xleftarrow{\$} \mathbb{Z}_q^*$ and defines $\widetilde{G}^{(I)} := [k^{(I)}]G$.

The garbler transmits to the evaluator the powers

$$\widetilde{G}^{(I)}_{-j} := [\Delta^{-j}]\widetilde{G}^{(I)} = [k^{(I)}\Delta^{-j}]G \qquad j \in \{0, 1, \ldots, I-1\},$$

including $\widetilde{G}^{(I)}_0 = \widetilde{G}^{(I)}$, but *not* any power $\widetilde{G}^{(I)}_{-j}$ for $j \geq I$. All tensor-product gates of arity I in the circuit share the same $\widetilde{G}^{(I)}$; the secrets $k^{(I)}$ for different arities are sampled independently.

The role of $k^{(I)}$ is to randomize the input to the random oracle X in a per-arity fashion. The verifier's view contains $\widetilde{G}^{(I)}_{-j}$ for $j \in \{0, \ldots, I-1\}$ but never $\widetilde{G}^{(I)}_{-j}$ for $j \geq I$; computing a withheld power from the transmitted ones is the Δ-power problem of Definition 41.

Definition 24. Let G be a gate with W_A, W_B as vectors of input wires, with respective nonces $R_{A,i}, R_{B,j}$ $(i \in \mathbb{Z}_n, j \in \mathbb{Z}_m)$. Let nonces $R_{O,k}$ for the output vector of wires W_O, which are defined (computed by garbler) as follows: $R_{O,k} = \prod_{i,j \,:\, f(i,j)=k} X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)})$, where $\widetilde{G}^{(2)}$ is the per-arity generator for $I = 2$ (Section 4.3.1).

Definition 25. Let the garbler compute

$$H = \Delta \prod_{i \in \mathbb{Z}_n, j \in \mathbb{Z}_m} X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)}) \tag{4}$$

and send it to the evaluator. The number H is called a *help* for the garbled function computation f. It only depends on the nonces of the input wires and on the secret $k^{(2)}$.

Lemma 7. *Let $\{Z_{ij}\}_{i,j}$ be independent uniform in $\mathbb{Z}_q^*$ and fix a hot index (a, b). Define $H := \Delta \cdot \prod_{i,j} Z_{ij}$ and $\widetilde{Z}_{ab} := H \cdot \big(\prod_{(i,j)\neq(a,b)} Z_{ij}\big)^{-1}$. Then $\widetilde{Z}_{ab} = \Delta \cdot Z_{ab}$, and, conditioned on $\{Z_{ij}\}_{(i,j)\neq(a,b)}$, the distribution of $\widetilde{Z}_{ab}$ is uniform in $\mathbb{Z}_q^*$.*

Proof. $H = \Delta\, Z_{ab} \cdot \prod_{(i,j)\neq(a,b)} Z_{ij}$, hence $\widetilde{Z}_{ab} = \Delta \cdot Z_{ab}$; multiplication by $\Delta \in \mathbb{Z}_q^*$ is a permutation of $\mathbb{Z}_q^*$. □

4.3.2 One-hot tensor product $T_{A\otimes B}$ of one-hot vectors of wires W_A, W_B

To obtain this auxiliary collection of Boolean wires, we need only to define its nonces. The index set of T is exactly the Cartesian product $A \times B$, hence we can use $(i, j) \in \mathbb{Z}_n \times \mathbb{Z}_m$ to indicate these entries.

Definition 26. Define the nonces of $T_{A\otimes B}$ as $R_{A\otimes B,(i,j)} = X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)})$, where $\widetilde{G}^{(2)}$ is the per-arity generator for $I = 2$ (Section 4.3.1). These are computed by the garbler since he knows the nonces of the input wires and $k^{(2)}$. Then

$$T_{A\otimes B}(x, y)_{ij} = R_{A\otimes B,(i,j)}\Delta^{\delta_i(x)\delta_j(y)} \qquad i \in \mathbb{Z}_n,\ j \in \mathbb{Z}_m.$$

Definition 27. For inputs $a \in \mathbb{Z}_n$ and $b \in \mathbb{Z}_m$, let $W_A(a), W_B(b)$ be their respective garblings on the input wires. For $(i, j) \neq (a, b)$ the evaluator computes first

$$w_{ij} = \begin{cases} X([W_A(a)_i\, W_B(b)_j]\widetilde{G}^{(2)}) & i \neq a \text{ and } j \neq b \\ X([W_A(a)_i\, W_B(b)_j]\widetilde{G}^{(2)}_{-1}) & (i = a) \text{ xor } (j = b) \end{cases}$$

Setting $w_{ab} = 1$, let $h = \prod_{i\in\mathbb{Z}_n, j\in\mathbb{Z}_m} w_{ij}$. Here $\widetilde{G}^{(2)}_{-1}$ is the per-arity generator power transmitted to the evaluator (Section 4.3.1).

Remark 19. Note that w_{ij} and h can be computed by the evaluator, since she holds the garbled wires $W_A(a), W_B(b)$ and she also knows the cleartexts a, b. On the other hand, the help H can only be computed by the garbler, who garbled the wires, but this help was previously sent to the evaluator.

Theorem 2. *We have*

$$T_{A\otimes B}(a,b)_{ij} = X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)})\Delta^{\delta_i(a)\delta_j(b)} = \begin{cases} w_{ij} & (i,j)\neq(a,b) \\ Hh^{-1} & i=a \text{ and } j=b \end{cases}.$$

Proof. Note that $w_{ij} = X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)}) = R_{A\otimes B,(i,j)} = T_{A\otimes B}(a,b)_{ij}$ for $(i,j)\neq(a,b)$, since $\widetilde{G}^{(2)}_{-1} = [\Delta^{-1}]\widetilde{G}^{(2)}$ in the mixed-hot case of w_{ij} cancels out the hot entries with Δ. On the other hand $Hh^{-1} = \Delta\, X([R_{A,a}R_{B,b}]\widetilde{G}^{(2)}) = R_{A\otimes B,(a,b)}\Delta = T_{A\otimes B}(a,b)_{ab}$. □

The following definition again is based on Lemma 3, replacing the sum with the product and with the addition of the elliptic curve which acts as a hashing function:

Definition 28. The one-hot garbled output of gate G computing f in terms of garbled one-hot inputs is defined as $\breve{f}(W_A(a), W_B(b))_k := \prod_{i\in\mathbb{Z}_n, j\in\mathbb{Z}_m \,:\, f(i,j)=k} T_{A\otimes B}(a,b)_{ij}$. The product over an empty set equals $1\in\mathbb{Z}_p$.

Remark 20. Note that at this point the evaluator, knowing a, b, $T_{A\otimes B}(a,b)$ and f, knows the cleartext $f(a,b)$ hence she knows the garbling of this output and the exact *position* $k = f(a,b)$ in this garbled wire where there is the hot entry Δ, so she can use this as the input of another garbled gate. Also note that with the auxiliary tensor $T_{A\otimes B}(a,b)$ she can obtain the output of any gate with the same garbled inputs $W_A(a), W_B(b)$.

Theorem 3. *The computation of $\breve{f}$ returns the one-hot output vector of wires evaluated at $f(a,b)$, that is: $W_O(f(a,b)) = \breve{f}(W_A(a), W_B(b))$.*

Proof. The proof is similar to that of Theorem 1: recalling Definition 24, if $k = f(a,b)$ then due to Theorem 2, both sides are equal to

$$\prod_{i\in\mathbb{Z}_n, j\in\mathbb{Z}_m \,:\, f(i,j)=f(a,b)} X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)})\Delta = R_{O,f(a,b)}\Delta = W_O(f(a,b))_{f(a,b)},$$

while for $k\neq f(a,b)$ both sides are equal to

$$\prod_{i\in\mathbb{Z}_n, j\in\mathbb{Z}_m \,:\, f(i,j)=k} X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)}) = R_{O,k} = W_O(f(a,b))_k. \qquad \square$$

Lemma 8. *If $f:\mathbb{Z}_n\times\mathbb{Z}_m\to\mathbb{Z}_d$ and $g:\mathbb{Z}_d\to\mathbb{Z}_r$, then $\widetilde{g\circ f} = \breve{g}\circ\breve{f}$.*

Proof. Identical to Lemma 6, with the tensor entries $T_{A\otimes B}(a,b)_{ij}$ in place of single-wire entries; the underlying set identity is the same disjoint-union decomposition of $\{(i,j) : g(f(i,j)) = \ell\}$ over the fibers of g. □

Remark 21. We point out that the composition properties *fail* when there is garbled tensor product *in the middle* of a garbling, that is, if the first gate computes $g:\mathbb{Z}_d\to\mathbb{Z}_{n_1}\times Z_{n_2}$ and $f:\mathbb{Z}_{n_1}\times\mathbb{Z}_{n_2}\to\mathbb{Z}_d$. To see this, note that $g = (g_1, g_2)$ with $g_i:\mathbb{Z}_d\to\mathbb{Z}_{n_i}$. Then the composition $f\circ g:\mathbb{Z}_d\to\mathbb{Z}_k$ is garbled for free (Section 4.2), with $\widetilde{(f\circ g)}(W_I(a))$, here W_I is the vector of input wires of the first gate, involving no tensor products neither elliptic curve operations, while $(\breve{f}\circ\breve{g})$ involves the garbled tensor product of $\breve{g_1}(W_I(a))$ and $\breve{g_2}(W_I(a))$.

Remark 22. This general case can be obtained similarly as we did in Section 4.3. Let $W_{A_0}, \ldots, W_{A_{I-1}}$ be the vectors of input wires of the gate computing f, with $A_j \subset \mathcal{W}$, and let $R_{A_j,i}$ for $j \in \mathbb{Z}_I$, $i \in \mathbb{Z}_{n_j}$ be the respective nonces. Let W_O be the output wires of the gate.

Definition 29. The nonces $R_{O,k}$ for W_O ($k \in \mathbb{Z}_m$) are defined (computed by the garbler) as $R_{O,k} = \prod_{\substack{i_j \in \mathbb{Z}_{n_j} \text{ s.t.} \\ f(i_0,\ldots,i_{I-1})=k}} X([\prod_{j\in\mathbb{Z}_I} R_{A_j,i_j}]\widetilde{G}^{(I)})$, where $\widetilde{G}^{(I)}$ is the per-arity generator for arity I (Section 4.3.1).

For this gate of arity I, the per-arity generator powers $\widetilde{G}^{(I)}_{-j} = [\Delta^{-j}]\widetilde{G}^{(I)}$ are transmitted to the evaluator for $j \in \{1, \ldots, I-1\}$, as described in Section 4.3.1. We note that $\widetilde{G}^{(I)}$ itself is *not* transmitted, nor is any power $\widetilde{G}^{(I)}_{-j}$ for $j \geq I$.

Now the computation of this gate proceeds as in the case of the two-variable function: provided the evaluator holds the actual values of the garbled input wires, that is $W_{A_j}(a_j)$, for $j \in \mathbb{Z}_I$ and $a_j \in \mathbb{Z}_{n_j}$, she now computes the tensor product

$$\bigotimes_{j\in\mathbb{Z}_I} W_{A_j}(a_j) = W_{A_0}(a_0) \otimes W_{A_1}(a_1) \otimes \cdots \otimes W_{A_{I-1}}(a_{I-1}).$$

With the help $H = \Delta \prod_{i_j \in \mathbb{Z}_{n_j}} X([R_{A_0,i_0} R_{A_1,i_1} \ldots R_{A_{I-1},i_{I-1}}]\widetilde{G}^{(I)})$, produced by the garbler and handed to the evaluator, she can produce the one-hot garbled tensor product $T_{\otimes j A_j}(a_0, \ldots, a_{I-1})$, and with that the computation of garbled f is defined as

$$\breve{f}(W_{A_0}(a_0), \ldots, W_{A_{I-1}}(a_{I-1}))_k := \prod_{\substack{i_j \in \mathbb{Z}_{n_j} \text{ s.t.} \\ f(i_0,\ldots,i_{I-1})=k}} T_{\otimes j A_j}(a_0, a_1, \ldots, a_{I-1})_{i_0 i_1 \ldots i_{I-1}} \quad k \in \mathbb{Z}_m.$$

We will not give the full details of the general case, but the reader should notice that again this is a well formed garbling: the output of this gate is exactly its output wire $W_O(z)$, evaluated at $z = f(a_0, \ldots, a_{I-1})$, that is

$$\breve{f}(W_{A_0}(a_0), \ldots, W_{A_{I-1}}(a_{I-1}))_k = R_{O,k} \cdot \Delta^{\delta_k(f(a_0,a_1,\ldots,a_{I-1}))} = W_O(f(a_0, \ldots, a_{I-1}))_k.$$

We will need a special case of this (with some modifications), when $n_i = 2$ for all i, regarding binary to word functions, see Section 4.3.4 below, which we will develop there in full detail.

4.3.3 Tuples of functions

When the output of f (defined in $\prod_j \mathbb{Z}_{n_j}$) is an ordered k-uple of words, its garbling can be dealt by just garbling each component $f_k : \prod_j \mathbb{Z}_{n_j} \to \mathbb{Z}_{m_k}$ with the process just described (as we did in Remark 21).

In this case the gate has input wires W_{A_j} ($j \in \mathbb{Z}_I$) and several output wires W_{O_k}. The following is the consistent definition of the output wire nonces for this gate, and it is just a generalization of the cases already discussed.

Definition 30. Let $R_{A_j,i}$ be the nonces for input wires W_{A_j} of gate G, then for $\ell \in \mathbb{Z}_{m_k}$, the nonces for the k-th output vector wire W_{O_k} are as in Definition 29

$$R_{O_k,\ell} := \prod_{\substack{i_j \in \mathbb{Z}_{n_j} \text{ s.t.} \\ f_k(i_0,\ldots,i_{I-1})=\ell}} X([\prod_{j\in\mathbb{Z}_I} R_{A_j,i_j}]\widetilde{G}^{(I)}).$$

With this definition, the computation of the garbled outputs of f can be obtained consistently, and

$$\breve{f}_k(W_{A_0}(a_0), \ldots W_{A_{I-1}}(a_{I-1}))_\ell = R_{O_k,\ell} \cdot \Delta^{\delta_\ell(f_k(a_0,a_1,\ldots,a_{I-1}))} = W_{O_k}(f(a_0, \ldots, a_{I-1}))_\ell.$$

4.3.4 Garbling binary-to-word functions and some applications

We want to deal now with a function $f : \mathbb{Z}_2^d \to \mathbb{Z}_n$. In this case we are then dealing with a d-variable function, each variable $z_i \in \{0,1\} = \mathbb{Z}_2$. At this point the evaluator holds, for this gate, the blinded generators $G_{-c} = [\Delta^{-c}k^{\mathsf{b}}]G$ for $c \in \{0,\dots,d-1\}$, where $k^{\mathsf{b}} \xleftarrow{\$} \mathbb{Z}_q^*$ is a per-gate secret (Algorithm 15); the power $[\Delta^{-d}k^{\mathsf{b}}]G$ is withheld. Throughout this section G_{-c} denotes these per-gate blinded generators and $\widetilde{G}^{\mathsf{b}} = [k^{\mathsf{b}}]G = G_0$. In this section we will need to use the original notation w_{i_k}, R_{i_k} for $i_k \in I$.

Remark 23. We note that for a bit z, we have $\delta_0(z) = 1 - z = \overline{z}$ and $\delta_1(z) = z$. Let $I = \{i_0, i_1\}$ and let $W(z) = (R_{i_0}\Delta^{\overline{z}}, R_{i_1}\Delta^z)$ be a one-hot vector of wires carrying only one bit (that is $z \in \mathbb{Z}_2 \simeq \{0,1\}$). By dropping the first entry we obtain a Boolean wire carrying the same information.

Definition 31. For a Boolean wire $w(z) = R\Delta^z$, we denote $r_1 = R, r_0 = (R\Delta)^{-1}$ and we build the *auxiliary one-hot vector* of Boolean wires

$$\mathcal{H}(z) = (r_0\Delta^{\delta_0(z)}, r_1\Delta^{\delta_1(z)}) = ((R\Delta)^{-1}\,\Delta^{\overline{z}}, R\,\Delta^z) = (w(z)^{-1}, w(z)) \qquad z \in \{0,1\}.$$

This is the analogue of the one-hot encoding of a bit $\mathcal{H} : \{0,1\} \mapsto \{0,1\}^2$ described in Definition 7.

1. $\mathcal{H}$ is hot in the first coordinate if and only if $z = 0$, and hot in the second if and only if $z = 1$: it is a one-hot vector of two Boolean wires, by Definition 3.2.(2).
2. If we drop the first entry of $\mathcal{H}$, we recover the original Boolean wire.
3. If we invert $\pmod p$ one of the wires of $\mathcal{H}$, we obtain the other wire.

Remark 24. In the process, we have obtained the following behaviour for not in our garbling scheme, when applied to a Boolean garbling: $\mathsf{not} : R\Delta^z \mapsto (R\Delta^z)^{-1}$. We leave to the reader to check that if we create $\mathcal{H}$ as in Definition 31 and drop the first entry of the one-hot output, we recover not as defined above.

Since notation is a bit heavy on this section, let us take a look at how things are for a bunch of Boolean wires:

Definition 32. For a set of indices $I = \{i_0,\dots,i_{d-1}\} \subset \mathcal{W}$, with $Z = z_0 \dots z_{d-1} \in \{0,1\}^d$ and a binary vector of wires

$$B_I(Z) = (w_{i_0}(Z),\dots,w_{i_{d-1}}(Z)) = (R_{i_0}\Delta^{z_0},\dots,R_{i_{d-1}}\Delta^{z_{d-1}})$$

we now have (for each $j \in \mathbb{Z}_d$) the auxiliary one-hot vectors of wires

$$\mathcal{H}_{i_j}(z) = (r_{i_j,0}\Delta^{\overline{z}}, r_{i_j,1}\Delta^z) = (w_{i_j}(z)^{-1}, w_{i_j}(z)) \qquad z \in \{0,1\},$$

with

$$r_{i_j,0} = (R_{i_j}\Delta)^{-1} \qquad r_{i_j,1} = R_{i_j} \qquad j \in \mathbb{Z}_d. \tag{5}$$

Let B_I be the input wires for the gate computing $f : \mathbb{Z}_2^d \to \mathbb{Z}_n$, with a binary entry ($d$-bits) and an arithmetic output. We will think of this gate as having d one-hot input wires $\mathcal{H}_{i_j}$ (one for each bit) and one-hot output wires W_O. Therefore we can apply our already defined algorithm to compute this gate (Section 22). Let us get into details:

Definition 33. The n nonces for the output wires W_O are obtained as before as $R_{O,\ell} := \prod_{\substack{z_j \in \{0,1\}\ s.t.\\ f(z_0,\dots,z_{d-1})=\ell}} X([\prod_{j\in\mathbb{Z}_d} r_{i_j,z_j}]\widetilde{G}^{\mathsf{b}})$.

For each $k \in \mathbb{Z}_{2^d}$ denote $\mathsf{bin}(k) = k[0] \dots k[d-1]$ as before, with $k[0]$ its LSB, then the help for this gate is

$$H = \Delta \prod_{k \in \mathbb{Z}_{2^{2d}}} X([\prod_{j \in \mathbb{Z}_d} r_{i_j,k[j]}]\widetilde{G}^{\mathsf{b}}) \tag{6}$$

computed by the garbler and passed on to the evaluator as circuit material.

Definition 34. We write $T_{\otimes\mathcal{H}} = T_{\otimes_{j\in\mathbb{Z}_d}\mathcal{H}_{i_j}}$ for the tensor product of the auxiliary input wires, to keep notation at bay. To obtain this auxiliary collection of Boolean wires, we need only to define its nonces. The index set of T is exactly $\{0,1\}^d$, hence we can use $\mathsf{bin}(k)$ for $k \in \mathbb{Z}_{2^d}$ to indicate these entries, and define the nonces of $T_{\otimes\mathcal{H}}$ as $R_{\otimes\mathcal{H},\,\mathsf{bin}(k)} = X([\prod_{j\in\mathbb{Z}_d} r_{i_j,k[j]}]\widetilde{G}^{\mathsf{b}})$ for $k \in \mathbb{Z}_{2^d}$. These are computed by the garbler since he knows the nonces of the input wires.

Definition 35. For fixed $a_0 a_1 \dots a_{d-1} \in \{0,1\}^d$, and by means of the flip operator T_* introduced in Definition 12, we consider the counting function

$$c(k) = \sum_{i=0}^{d-1} (T_{\mathsf{bin}(k)} a_0 a_1 \dots a_{d-1})[i] \qquad k \in \mathbb{Z}_{2^d}. \tag{7}$$

This function returns the number of hot entries (equivalently the number of times Δ appears) in the tensor product of the auxiliary one-hot garblings

$$(\bigotimes_{j=0}^{d-1} \mathcal{H}_{i_j}(a_j))_{\mathsf{bin}(k)} = \prod_{j=0}^{d-1} \mathcal{H}_{i_j}(a_j)_{k[j]} = \prod_{i=0}^{d-1} w_{i_j}(a_j)^{2k[j]-1}.$$

Definition 36. Let $w_{i_j} = w_{i_j}(a_j)$ be the binary garbling of cleartext $a_0 \dots a_{d-1} \in \{0,1\}^d$, in the input wires of the gate computing $f : \mathbb{Z}_2^d \to \mathbb{Z}_n$. We first produce their garbled tensor product: for each multi-index $\mathsf{bin}(k) \in \{0,1\}^d$, $k \in \mathbb{Z}_{2^d}$, we let

$$w_k = \begin{cases} X([\prod_{j\in\mathbb{Z}_d} w_{i_j}^{2k[j]-1}]\, G_{-c(k)}) & \mathsf{bin}(k) \neq a_0 \dots a_{d-1} \\ 1 & \mathsf{bin}(k) = a_0 \dots a_{d-1} \end{cases}$$

Note that the evaluator, in possession of the G_c and the w_{i_j}, can compute all of the w_k and she can also compute $h = \prod_{k\in\mathbb{Z}_{2^d}} w_k$. With a similar proof that of Theorem 2, it follows that

$$T_{\otimes\mathcal{H}}(a_0 \dots a_{d-1})_{\mathsf{bin}(k)} = \begin{cases} w_k & \mathsf{bin}(k) \neq a_0 \dots a_{d-1} \\ Hh^{-1} & \mathsf{bin}(k) = a_0 \dots a_{d-1} \end{cases}$$

A key observation for this proof is that

$$\begin{array}{ll} w_k = X([\prod_{j=0}^{d-1} r_{i_j,k[j]}]\widetilde{G}^{\mathsf{b}}) & \mathsf{bin}(k) \neq a_0 \dots a_{d-1} \\ Hh^{-1} = \Delta \cdot X([\prod_{j=0}^{d-1} r_{i_j,a_j}]\widetilde{G}^{\mathsf{b}}) & \mathsf{bin}(k) = a_0 \dots a_{d-1} \end{array}.$$

Definition 37. Let

$$B = B_I(a_0, \dots, a_{d-1}) = (w_{i_0}, \dots, w_{i_{d-1}})$$

be the binary garbling of the input wires B_I. For each $\ell \in \mathbb{Z}_n$ we let

$$\breve{f}(B)_\ell = \prod_{k\in\mathbb{Z}^{2^d}\,:\, f(\mathsf{bin}(k))=\ell} T_{\otimes\mathcal{H}}(a_0, \dots, a_{d-1})_{\mathsf{bin}(k)}$$

where as before, the product over the empty set is defined as $1 \in \mathbb{Z}_p$.

We leave to the reader to verify the consistency, that is if $Z = z_0 \cdots z_{d-1}$ then

$$\check{f}(B_I(Z))_\ell = R_{O,\ell}\, \Delta^{\delta_\ell(f(Z))} = W_O(f(Z))_\ell.$$

As an application, we now use this to translate a binary garbled wire to a one-hot arithmetic garbled wire.

4.4 Algorithms for translation and more applications

We now describe how to consistently transform garblings of some type (word, binary, one-hot) into the other two types of garblings. We begin with a direct application of the results of the previous section.

4.4.1 Binary to one-hot: bin2hot

Inputs a binary garbling and outputs a one-hot garbling of the same cleartext (Algorithm 15 in the next section). This is an overview with some remarks: taking $n = 2^d$, consider $f : \mathbb{Z}_2^d \to \mathbb{Z}_n$ given by $z_0 z_1 \dots z_{d-1} \mapsto \sum_{j=0}^{d-1} z_j\, 2^j$.

The garbling of this function is a particular instance of the general discussion of the previous section. Clearly $f(\mathsf{bin}(k)) = \ell$ if, and only if, $k = \ell$, and if

$$z_0 z_1 \dots z_{d-1} = a[0] \dots a[d-1] = \mathsf{bin}(a),$$

then $f(z_0, \dots, z_{d-1}) = a$. Therefore if $B_I(a[0] \dots a[d-1]) = (w_{i_0}, \dots, w_{i_{d-1}})$ is a garbling of $\mathsf{bin}(a)$, the one-hot garbled output $W(a = f(\mathsf{bin}(a)))$ of bin2hot is

$$\mathsf{bin2hot} : B_I(\mathsf{bin}(a)) \mapsto W(a)$$

for $W(a)_k = X([\prod_{j=0}^{d-1} w_{i_j}^{2k[j]-1}]\, \widetilde{G}^{\mathsf{b}}_{-c(k)})$ $(k \neq a)$, $W(a)_a = H \prod_{k \neq a} W(a)_k^{-1}$, with H derived from (6) and (5) as $H = \Delta \prod_{k \in \mathbb{Z}_{2^d}} X([\prod_{j \in \mathbb{Z}_d} R_{i_j}^{2k[j]-1} \Delta^{k[j]-1}]\, \widetilde{G}^{\mathsf{b}})$.

4.4.2 One-hot to binary: hot2bin

The inverse of the previous primitive bin2hot can be obtained combining previous ideas, and it is then computed "for free" (no material transmitted). As usual $d = \lceil \log_2 n \rceil$. We have to think of the operation that sends a one-hot cleartext to a binary cleartext as a collection of mappings $f_j : \mathbb{Z}_n \to \mathbb{Z}_2$ $(j \in \mathbb{Z}_d)$ defined as $f_j[k] = k[j]$, where $k[0] \cdots k[d-1] = \mathsf{bin}(k)$ for $k \in \mathbb{Z}_n$. This gate has then a one-hot vector of input wires W_I (length n) and d Boolean output wires w_{i_j}, $j \in \mathbb{Z}_d$. Hence denoting

$$\mathsf{hot2bin} : W_I(a) \mapsto B_O(\mathsf{bin}(a)) = (w_{i_0}, \dots, w_{i_{d-1}})$$

and looking at Section 4.2 we have

$$w_{i_j} = \prod_{k \in \mathbb{Z}_n \,:\, f_j(k)=1} W_I(a)_k = \prod_{k \in \mathbb{Z}_n \,:\, k[j]=1} W_I(a)_k \qquad j \in \mathbb{Z}_d.$$

4.4.3 Binary to word: bin2word

Inputs a binary garbling $B_I(a)$, outputs a word garbling $\Omega(a)$ of the same cleartext, for free. If the nonces of the input wires are R_{i_j}, we have

$$B_I(a) = (w_{i_0}, \dots, w_{i_{d-1}}) = (R_{i_0} \Delta^{a[0]}, \dots, R_{i_{d-1}} \Delta^{a[d-1]}),$$

the nonce of the output wire is obtained as $r = \prod_{j=0}^{d-1} R_{i_j}^{2^j}$, and we compute

$$\prod_{j=0}^{d-1} w_{i_j}^{2^j} = \prod_{j=0}^{d-1} (R_{i_j} \Delta^{a[j]})^{2^j} = \prod_{j=0}^{d-1} R_{i_j}^{2^j} \cdot \Delta^a = r\Delta^a$$

obtaining a word garbling of a, that is $\mathsf{bin2word} : (w_{i_0}, \ldots, w_{i_{d-1}}) \mapsto \prod_{j=0}^{d-1} w_{i_j}^{2^j} = \Omega(a)$.

4.4.4 Word to one-hot: word2hot

Inputs a word garbling, outputs a one-hot garbling of the same cleartext $a \in \mathbb{Z}_n$.

A different generator G of the curve is needed for this algorithm, that is to securely compute it is necessary that the evaluator does not know the generator of the curve. The garbler uses the circuit-wide secret $K \in \mathbb{Z}_q^*$ sampled by Gb (Algorithm 6), produces the generator $G' = [K]G$, and gives $G'_i = [\Delta^i]G'$ to the evaluator, $-(n-1) \le i \le n-1$, $i \neq 0$. If the (word) input wire has nonce r, and W_I are the output wires with $I = \{i_0, \ldots, i_{n-1}\}$, then $R_{i_j} = X([r\Delta^j]G') \qquad j \in \mathbb{Z}_n$ are by definition the nonces for the output wire. Then for any $a \in \mathbb{Z}_n$ and its word garbling $\Omega(a) = r\Delta^a$ in the input wire, the evaluator can compute $[\Omega(a)]G'_c = [r\Delta^{a+c}]G'$ thus obtaining:

$$X([\Omega(a)]G'_{j-a}) = X([r\Delta^j]G') = W_I(a)_j \quad j \neq a \qquad \text{and} \qquad h = \prod_{j \in \mathbb{Z}_n : j \neq a} W_I(a)_j.$$

On the other hand the garbler provides the help

$$H = \Delta \prod_{j=0}^{n-1} X([r\Delta^j]G') \tag{8}$$

to the evaluator and with it she computes $Hh^{-1} = \Delta\, X([r\Delta^a]G') = W_I(a)_a$, obtaining consistently the full one-hot output garbled wire $W_I(a) = \mathsf{word2hot}(\Omega(a))$. Remark 18 applies to all these computations.

4.4.5 One-hot to word: hot2word

Inputs a one-hot garbling, outputs a word garbling of the same cleartext, for free. If the input wires W_I have nonces R_{i_j} $(j \in \mathbb{Z}_n)$, the output wire has nonce $r = \prod_j R_{i_j}^j$. Then we define $\mathsf{hot2word} : W_I(a) \mapsto \prod_{j \in \mathbb{Z}_n} (W_I(a)_j)^j = \prod_j R_{i_j}^j \Delta^a := r\Delta^a = \Omega(a)$ which implements the functionality.

4.4.6 Word to bin: word2bin

Inputs a word garbling, outputs a binary garbling. It is implemented by composition: $\mathsf{word2bin}(z) = \mathsf{hot2bin}(\mathsf{word2hot}(z))$.

It costs $1 + 0 = 1$ ciphertexts of material: for each output bit j, the evaluator groups the word2hot entries $X([\Omega(a)]G'_{k-a})$ over $\{k : k[j] = 1\}$, using Hh^{-1} (help (8)) for the hot factor when $a[j] = 1$; this is a direct application of the composition rule and of §4.4.4. Again, Remark 18 applies to this construction.

4.4.7 More applications: sum and product of integers

For $f = +$ the sum of two integers modulo n, the truth table of f is a $n \times n$ symmetric matrix, with $(i, k-i) \mapsto k$ for $i \in \mathbb{Z}_n$. Then by the results of Section 4.3 we have

$$\breve{+}(W_A(a), W_B(b))_k := \prod_{i \in \mathbb{Z}_n} T_{A \otimes B}(a, b)_{i,k-i} \qquad k \in \mathbb{Z}_n.$$

For $f = \times$ the product of two integers modulo n, with n prime, for $k \neq 0$ we have $(i, ki^{-1}) \mapsto k$ for $i \neq 0$ while $(0, j) \mapsto 0$ and $(i, 0) \mapsto 0$. Hence

$$\check{\times}(W_A(a), W_B(b))_k := \prod_{i \neq 0} T_{A\otimes B}(a, b)_{i,ki^{-1}} \qquad k \neq 0$$

while $\check{\times}(W_A(a), W_B(b))_0 := \prod_{i,j \in \mathbb{Z}_n} T_{A\otimes B}(a, b)_{i,0} T_{A\otimes B}(a, b)_{0,j}$.

Remark 25. For large integers $n, m \simeq 2^{32}$, these operations are impractical, since the tensor product involves $n \times m$ products that is $\simeq 2^{64}$ elliptic curve multiplications. A possible approach is to take the discrete logarithm, operate with word garblings and go back to one-hot, what follows is a description of this approach.

Let W_A, W_B be the arrays of wires carrying a, b that we want to multiply. Pick the Mersenne prime $m = 2^{31} - 1$, and a generator g of $\mathbb{Z}_m^*$ (say $g = 2$)

1. Compute the table of values of the discrete logarithm function $l : \mathbb{Z}_m^* \to \mathbb{Z}_m$, $2^{l(x)} = x \pmod m$. Define $l(0) = 2m - 3$ to avoid collisions, since $2^{2m-3} \equiv 2^{-1} \equiv 2^{m-2} \pmod m$ and all the other discrete logarithms are in $[0, m - 2]$.

2. Apply the basic functionality of Section 4.2 with the discrete logarithm function l to our garbled inputs $W_A(a)$ and $W_B(b)$; the output is $W_O(l(a)), W_O(l_(b))$.

3. Apply the hot2word functionality (Section 4.4.5) to both of them to obtain word garblings of the logarithms $\Omega_l(l(a)) = R\Delta^{l(a)}, \Omega_l(l(b)) = R\Delta^{l(b)}$.

4. Multiply in $\mathbb{Z}_q^*$ to obtain $\Omega(\sigma) = \Omega_l(l(a))\,\Omega_l(l(b)) = r_A r_B\,\Delta^\sigma$, a word garbling of the *unreduced* exponent $\sigma := l(a) + l(b) \in [0,\, 2(2m - 3)]$; note $\sigma \equiv l(ab) \pmod{m-1}$ when $a, b \neq 0$.

5. Apply word2hot over $\mathbb{Z}_{4m-5}$ (Section 4.4.4) to obtain $W_P(\sigma)$ with $P = \{i_0, \ldots, i_{4m-6}\}$.

6. Let $e : \mathbb{Z}_{4m-5} \to \mathbb{Z}_m$ be given by $e(x) = 2^x \bmod m$ for $x \leq 2m - 4$ and $e(x) = 0$ for $x \geq 2m - 3$; we again apply the basic functionality of Section 4.2 and we end up with $W_O(ab)$, a valid garbling of the product ab.

Remark 26. The functionalities of steps (2), (3), (4), (6) are for free, while the one in step (5) requires one ciphertext. Regarding the number of operations, it is within $O(m)$. Step (2) can be done by brute force given the size of the numbers.

5 Security

This section begins by proving a detailed security proof for the authenticity property of the One Hot Modular Garbling (OHMG) scheme within the Universal Composability (UC) framework. The proofs address adaptive corruption of the prover for the stand-alone gate protocols (Remark 41) and static corruption for the commitment-bearing protocols (Remarks 40 and 38), and rest on the Δ-power assumptions of Definitions 41 and 42. Since the garbling scheme is modular by design, it will suffice to prove the authenticity property for each functionality. This tackles the security in the corrupt verifier/honest prover scenario. To finish the paper, we will dedicate the last section to prove security in the honest verifier/corrupt prover scenario.

Remark 27. Throughout this section, the UC security analysis makes use of the global setup introduced in Section 5.2 with $\mathcal{G}_{\mathsf{trust}}$, and $\mathcal{G}_{\mathsf{RO}}^X$.

Remark 28. Our security analysis is conducted in the GUC-RO model [CDPW07; CJS14]. Chevalier and Sageloli [CS26] recently identified definitional gaps in Plain UC [Can20] affecting the standard UC Theorem and proposed an updated framework. All functionalities and protocols in this paper satisfy the corresponding hypotheses of Updated UC: ideal functionalities are sub-party respecting under the new extended-session definition; all functionalities and protocols are sid-exposing, since sid is an explicit component of every ITI identity and every operation request to $\mathcal{G}_{\mathsf{trust}}$ and $\mathcal{G}_{\mathsf{RO}}^{X}$; and each of our security theorems compares a real protocol π to an ideal protocol $\mathrm{IDEAL}_{\mathcal{F}}$ with pairwise disjoint panoptic sessions by code separation. We follow the standard default-reject convention [Can20, §3.1.2]: any message arriving at a protocol ITI of π_f, π_{TP}, or π_{GC} that is not handled by an explicit case in the algorithm, in particular, backdoor messages and inputs from ITIs outside the corresponding panoptic session, is silently rejected. A formal extension of these guarantees to GUC, mirroring the Plain UC update of [CS26], remains open; we rely on [CDPW07] and [CJS14] for the GUC-RO instantiation.

5.1 UC authenticity in the honest prover, corrupt verifier scenario

Definition 38. The UC framework compares a *real-world* execution of a protocol π run by interactive parties (for instance prover P and evaluator V) with an *ideal-world* execution in which a trusted party implements an ideal functionality $\mathcal{F}$. The environment $\mathcal{Z}$ interacts with the parties and with a real-world adversary $\mathcal{A}$ or an ideal-world simulator Sim, and may adaptively corrupt parties during execution, revealing their internal states.

Formalized by Canetti [Can20], the UC paradigm is the cornerstone of modular security proofs. We use it in the standard way, specialized to our two-party setting.

Definition 39. Let $\mathcal{F}$ be an ideal functionality and let π be a protocol with the same party structure, in the $(\mathcal{G}_{\mathsf{trust}}, \mathcal{G}_{\mathsf{RO}}^{X})$-hybrid model. We consider PPT environments $\mathcal{Z}$ and PPT adversaries $\mathcal{A}$, allowing adaptive corruptions unless a statement explicitly restricts the corruption model, in a model without secure erasures. If there exists a PPT simulator Sim such that for all PPT $\mathcal{A}, \mathcal{Z}$ the ensembles $\mathsf{REAL}_{\mathcal{A},\pi}(\mathcal{Z})$ and $\mathsf{IDEAL}_{\mathsf{Sim},\mathcal{F}}(\mathcal{Z})$ are computationally indistinguishable, then π securely realizes $\mathcal{F}$ in the UC sense. The concrete security property achieved by π is determined by the behaviour of $\mathcal{F}$.

Lemma 9. *If $\mathcal{F}$ rejects every string other than its recorded output on its* Verify *interface, π UC-realizes $\mathcal{F}^{-\mathsf{Verify}}$ (i.e. $\mathcal{F}$ without that interface), and the acceptance check of the authenticity experiment is efficiently computable by $\mathcal{Z}$ in the real execution (an* adjudication predicate*), then π satisfies authenticity in the sense of Definition 4.*

Proof. An adversary breaking authenticity against π with non-negligible probability yields an environment that runs the authenticity experiment inside the UC execution, detects success via the adjudication predicate, and outputs 1 on a successful forgery. Its output is 1 with non-negligible probability in the real execution and with probability 0 in the ideal one, contradicting UC realization. □

Definition 40. Recall that $E(\mathbb{Z}_p)$ is an elliptic curve, and $\mathbb{G}$ is the multiplicative group generated by some generator $G \in \mathbb{G}$. We state here our main assumptions.

1. (ECDLP) For G of prime order $q \approx 2^{\kappa}$, the elliptic-curve discrete logarithm problem is hard: $\mathsf{Adv}_{\mathsf{DL}}(\kappa) := \Pr[\mathcal{A}_{\mathsf{DL}}(G, [k]G) = k] \leq \varepsilon_{\mathsf{DL}}(\kappa)$, matching the best known generic attack (Pollard's rho [HMV04], time $O(\sqrt{q}) = O(2^{\kappa/2})$; no subexponential attack is known for well-chosen curves). For secp256k1, $\varepsilon_{\mathsf{DL}} \approx 2^{-\kappa/2}$.

2. (Random oracle for X) The masking map X is modelled as the global random oracle $\mathcal{G}_{\mathsf{RO}}^{X}$ of Algorithm 8, with codomain $\mathbb{Z}_q^*$; a distinguisher making q_X queries has advantage $\leq q_X/q$.

3. (Hash) $\mathsf{Hash} : \mathbb{Z}_p \to \{0,1\}^\lambda$, $\lambda \geq 2\kappa$, is collision-resistant and preimage-resistant, with advantages $\varepsilon_{\mathsf{cr}}, \varepsilon_{\mathsf{pre}} = \mathsf{negl}(\kappa)$. ($\mathsf{Hash}$ is not a random oracle.)

4. (Δ-power hardness) The ΔP-DH family of Definition 41 is hard for the index sets used by OHMG.

Definition 41 (Δ-Power Diffie-Hellman family)**.** Let $\langle G \rangle$ have prime order q. Sample $k, \Delta \xleftarrow{\$} \mathbb{Z}_q^*$. For a finite index set $S \subset \mathbb{Z}$ and a target $t \in \mathbb{Z} \setminus S$, the (S,t)-Δ*P-DH problem* is: given $\big(G,\ \{[k\Delta^s]G\}_{s\in S}\big)$, compute $[k\Delta^t]G$. We assume that for the index sets arising in OHMG this problem is hard for every PPT algorithm, with advantage $\mathsf{Adv}^{\Delta\mathsf{P}}_{S,t}(\kappa) = \mathsf{negl}(\kappa)$:

- arity-I tensor: $S = \{0, -1, \ldots, -(I-1)\}$, $t = -I$;
- `word2hot` of length n: $S = \{\pm 1, \ldots, \pm(n-1)\}$, $t = 0$ (the missing middle power).

For $I = 2$ the tensor instance ($S = \{0,-1\}$, $t = -2$) is exactly Square-CDH relative to base $\widetilde{G}_0^{(2)}$ and ratio Δ^{-1}, which is equivalent to CDH [MW99]; the general and missing-middle instances are q-type Diffie-Hellman-exponent assumptions in the sense of [BGW05].

Definition 42 (Joint OHMG-power assumption)**.** Sample $\Delta \xleftarrow{\$} \mathbb{Z}_q^*$ and independent blinding scalars $k^{(2)}, \ldots, k^{(I_{\max})}, K, k_1^{\mathsf{b}}, \ldots, k_l^{\mathsf{b}} \xleftarrow{\$} \mathbb{Z}_q^*$, and let Mat be the full list of group elements transmitted by Gb: $\{[k^{(I)}\Delta^{-j}]G\}_{2\leq I\leq I_{\max},\, 0\leq j\leq I-1}$, $\{[K\Delta^i]G\}_{i\in\{\pm1,\ldots,\pm(n-1)\}}$, and $\{[k_\iota^{\mathsf{b}}\Delta^{-c}]G\}_{\iota,\, 0\leq c\leq d_\iota-1}$. The *joint OHMG-power problem* is: given (G, Mat), output any withheld element, that is $[k^{(I)}\Delta^{-I}]G$ for some I, $[K]G$, or $[k_\iota^{\mathsf{b}}\Delta^{-d_\iota}]G$ for some ι. We assume this problem is hard, with advantage $\mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) = \mathsf{negl}(\kappa)$. Each single-family instance above is the special case in which Mat contains one family; in the generic group model the joint problem inherits the $O(Q^2/q)$ bound of the q-type Diffie-Hellman-exponent framework of [BGW05], since every transmitted scalar is a distinct Laurent monomial in the formal variables $(\Delta, k^{(2)}, \ldots, K, k_1^{\mathsf{b}}, \ldots)$ and every withheld target lies outside their linear span.

Proposition 1 (Generic hardness of the joint OHMG-power problem)**.** *Let* Mat *be as in Definition 42, over a group* $\langle G \rangle$ *of prime order* q*. Let* $M := |\mathsf{Mat}| + 1$ *count the given elements (including* G*), let* $\tau := (I_{\max} - 1) + 1 + l$ *count the withheld targets, and set*

$$e := \max\big(I_{\max},\ \max_{\iota} d_\iota,\ n-1\big), \qquad D := n + e.$$

In the generic group model, every adversary that performs at most Q *group operations solves the joint OHMG-power problem with probability at most*

$$\mathsf{Adv}^{\Delta\mathsf{P},\mathrm{gen}}(Q) \ \leq\ \frac{(Q+M+\tau+1)^2\, D}{2\,(q-1)} \ +\ \frac{1}{q-Q-M} \ =\ O\Big(\frac{Q^2 D}{q}\Big).$$

Proof. Treat the secret scalars as formal variables; every element of $\{G\} \cup \mathsf{Mat}$ then has a distinct Laurent monomial as discrete logarithm, and no withheld target occurs among them: within a family the Δ-degrees differ, across families the variable supports differ. A generic adversary (Shoup's model [Sho97]) only forms $\mathbb{Z}_q$-linear combinations of the given monomials, and distinct Laurent monomials are linearly independent over $\mathbb{Z}_q$, so no tracked polynomial ever equals a target formally. The advantage is therefore bounded by the probability that two distinct tracked Laurent polynomials collide at the random point: clearing denominators by Δ^e leaves nonzero polynomials of total degree at most D, and the Schwartz-Zippel lemma over $\mathbb{Z}_q^*$, with a union bound over at most $\binom{Q+M+\tau+1}{2}$ pairs and the probability at most $1/(q-Q-M)$ of naming an unseen encoding, gives the stated bound; this is the plain-group instance of the master-theorem template of [BGW05]. □

Remark 29. Three comments are in order. (i) The bound explains why joint hardness is plausible despite the heavy correlation of Mat: all correlations among the transmitted elements are *multiplicative* relations between the underlying monomials, whereas a generic adversary is confined to the $\mathbb{Z}_q$*-linear* span of the given exponents, which the targets avoid. This is the plain-group specialization of the master theorem for the general Diffie-Hellman exponent framework [BGW05], matching the remark after Definition 41; the $I = 2$ single-family case remains equivalent to CDH [MW99], which serves as a sanity check that the family is not generically degenerate. (ii) The generic bound is a heuristic: it rules out attacks that use the group operation as a black box, and says nothing about attacks exploiting the concrete structure of secp256k1; this is the same caveat that already applies to Definition 41. (iii) Definition 42 follows Algorithm 6 in using a single circuit-wide K for `word2hot`; if instead a fresh K_ω is sampled per gadget (as in Algorithm 16), Mat acquires one family $\{[K_\omega \Delta^i]G\}_i$ per gadget, with one fresh formal variable each, and the proposition holds verbatim with the correspondingly larger M and τ.

5.2 Global model and setup

We use two global ideal functionalities shared by all sessions: (i) a public bulletin board $\mathcal{G}_{\mathsf{trust}}$ to store commitments for input/output wires and public metadata; (ii) a programmable random oracle for X, denoted $\mathcal{G}_{\mathsf{RO}}^{X}$. All calls include a session identifier sid. Calls to X also include a domain-separation tag tag describing the component and role. Given a map BB, we introduce the following set: $\mathrm{dom}(\mathsf{BB}) := \{(\mathsf{sid}, \mathsf{key}) \mid \mathsf{BB}[\mathsf{sid}, \mathsf{key}] \text{ is defined}\}$.

Definition 43. A *valid key* for session sid is a tuple of one of the following forms:

1. **Input wire commitment:** $\mathsf{key} = (\mathsf{pid}, C, \mathsf{in}, i, \sigma)$ where $\mathsf{pid} \in \mathcal{P}$ is a party identifier, C is a circuit identifier, $i \in \mathbb{N}$ is a wire index, and $\sigma \in \{0, 1\}$ is the label bit.

2. **Output wire commitment:** $\mathsf{key} = (\mathsf{pid}, C, \mathsf{out}, o, \tau)$ where $\mathsf{pid} \in \mathcal{P}$, C is a circuit identifier, $o \in \mathbb{N}$ is an output wire index, and $\tau \in \{0, 1\}$ is the label bit.

3. **Auxiliary commitment:** $\mathsf{key} = (\mathsf{pid}, C, \mathsf{aux}, \mathsf{tag})$ where $\mathsf{pid} \in \mathcal{P}$, C is a circuit identifier, and $\mathsf{tag} \in \{0, 1\}^*$ is an application-specific tag.

We write $\mathsf{owner}(\mathsf{key})$ for the first component pid of a valid key, and $\mathsf{Valid}(\mathsf{sid}, \mathsf{key})$ for the predicate that returns true if, and only if, key has one of the above forms.

Algorithm 7 Global bulletin board $\mathcal{G}_{\mathsf{trust}}$

1: **State:** A map $\mathsf{BB} : (\mathsf{sid}, \mathsf{key}) \mapsto (\mathsf{val}, \mathsf{time})$, initially empty. A monotonic counter $\mathsf{clock} \leftarrow 0$.
2: **Put:** On input $(\mathsf{Put}, \mathsf{sid}, \mathsf{key}, \mathsf{val})$ from party P:
   (a) **Validity check:** If $\neg\mathsf{Valid}(\mathsf{sid}, \mathsf{key})$, output $(\mathsf{Error}, \mathsf{sid}, \texttt{INVALID_KEY})$ to P and halt.
   (b) **Ownership check:** If $\mathsf{owner}(\mathsf{key}) \neq \mathsf{id}(P)$, output $(\mathsf{Error}, \mathsf{sid}, \texttt{NOT_OWNER})$ to P and halt.
   (c) **First-write-wins:** If $(\mathsf{sid}, \mathsf{key}) \in \mathrm{dom}(\mathsf{BB})$, output $(\mathsf{Error}, \mathsf{sid}, \texttt{ALREADY_SET})$ to P and halt.
   (d) **Commit:** Set $\mathsf{clock} \leftarrow \mathsf{clock} + 1$. Set $\mathsf{BB}[\mathsf{sid}, \mathsf{key}] \leftarrow (\mathsf{val}, \mathsf{clock})$. Broadcast $(\mathsf{Commit}, \mathsf{sid}, \mathsf{key}, \mathsf{clock})$ to all parties. Output $(\mathsf{Ack}, \mathsf{sid}, \mathsf{key}, \mathsf{clock})$ to P.
3: **Get:** On input $(\mathsf{Get}, \mathsf{sid}, \mathsf{key})$ from any party:
   (a) If $(\mathsf{sid}, \mathsf{key}) \in \mathrm{dom}(\mathsf{BB})$, let $(\mathsf{val}, t) \leftarrow \mathsf{BB}[\mathsf{sid}, \mathsf{key}]$ and output $(\mathsf{Value}, \mathsf{sid}, \mathsf{key}, \mathsf{val}, t)$.
   (b) Otherwise, output $(\mathsf{Value}, \mathsf{sid}, \mathsf{key}, \bot, \bot)$.
4: **List:** On input $(\mathsf{List}, \mathsf{sid})$ from any party, output the set $\{(\mathsf{key}, \mathsf{val}, t) : \mathsf{BB}[\mathsf{sid}, \mathsf{key}] = (\mathsf{val}, t)\}$.
5: **Clock:** On input $(\mathsf{Clock}, \mathsf{sid})$ from any party, output $(\mathsf{Time}, \mathsf{sid}, \mathsf{clock})$.

Remark 30. Algorithm 7 enforces writer-binding via the ownership check in step 2(b): a Put operation for key key succeeds only if the calling party P satisfies $\mathsf{id}(P) = \mathsf{owner}(\mathsf{key})$. This ensures that:

1. No party can write commitments under another party's namespace.
2. The prover P with identity pid has exclusive write access to all keys of the form $(\mathsf{pid}, \cdot, \cdot, \cdot)$.
3. Once a key is written (first-write-wins), no party (including the owner) can modify or overwrite it.

The validity check in step 2(a) additionally ensures that only well-formed keys (per Definition 43) are accepted, preventing injection of malformed metadata.

Remark 31. We model $\mathcal{G}_{\mathsf{trust}}$ as a *global setup* in the sense of Canetti et al. [CDPW07]: it is shared across all protocol sessions and is directly accessible by the environment $\mathcal{Z}$. This is essential for our application, where commitments posted in one session (such as the garbling phase) must be readable in another (e.g., the verification phase). The global nature of $\mathcal{G}_{\mathsf{trust}}$ is reflected in the session identifier sid being an explicit parameter, allowing the environment to correlate entries across sessions.

Remark 32. The functionality $\mathcal{G}_{\mathsf{trust}}$ should be viewed as an abstraction of an append-only, publicly readable ledger. We only rely on its ledger-like properties (global visibility, first-write-wins, and immutability of posted entries) and do not model the underlying consensus protocol. This follows the common approach of modelling blockchains as global ledger functionalities in UC-style analyses [KMS+16]. All our UC security theorems are therefore relative to the assumption that the deployment provides such a ledger functionality.

Algorithm 8 Global programmable random oracle for X

1: **State:** a table $\mathsf{T} : \{0,1\}^* \to \mathbb{Z}_q^*$, initially empty. The encoding function $\mathsf{enc} : \{0,1\}^* \times \{0,1\}^* \times \mathbb{G} \to \{0,1\}^*$ is any prefix-free injective encoding.
2: **Query:** On $(\mathsf{Query}, \mathsf{sid}, \mathsf{tag}, U)$ where $U \in \mathbb{G}$ is canonically encoded:
- Let $x \leftarrow \mathsf{enc}(\mathsf{sid}, \mathsf{tag}, U)$.
- If $x \notin \mathrm{dom}(\mathsf{T})$, sample $y \leftarrow \mathbb{Z}_q^*$ and set $\mathsf{T}[x] \leftarrow y$.
- Output $(\mathsf{Ans}, \mathsf{sid}, \mathsf{tag}, U, \mathsf{T}[x])$.

3: **Program:** On $(\mathsf{Program}, \mathsf{sid}, \mathsf{tag}, U, y)$ from an ITI M: if M is not the adversary (or simulator) of session sid, ignore the request. Otherwise, with $x = \mathsf{enc}(\mathsf{sid}, \mathsf{tag}, U)$ and $x \notin \mathrm{dom}(\mathsf{T})$, set $\mathsf{T}[x] \leftarrow y$. (Restricted programmability in the sense of [CDG+18]; adversarial real-world programming at honestly generated points is accounted for in Lemma 12.)

Remark 33. In the real-world execution, honest parties and the adversary only have access to the Query interface of $\mathcal{G}_{\mathsf{RO}}^X$. The $\mathsf{Program}$ interface is reserved exclusively for the ideal-world simulator in our UC proofs, and is never called by real-world code. This matches the standard programmable random-oracle modelling in the GUC-RO literature: programmability is a power of the simulator, not of real-world parties.

Remark 34. For the elliptic curve secp256k1 with generator G of prime order $q \approx 2^{256}$, we assume the ECDLP is hard: for any PPT algorithm $\mathcal{A}_{\mathsf{DL}}$, $\mathsf{Adv}_{\mathsf{DL}}(\kappa) := \Pr[\mathcal{A}_{\mathsf{DL}}(G, [K]G) = K] \le 2^{-\kappa/2}$. This bound matches the best known generic attack (Pollard's Rho [HMV04]), which has time complexity $O(\sqrt{q}) = O(2^{\kappa/2})$.

Algorithm 9 Protocol π_f: Garbling and Evaluation of $f : \mathbb{Z}_n \to \mathbb{Z}_m$ in OHMG

Require: Security parameter κ, prime $p \approx 2^\kappa$, circuit C with gate computing $f : \mathbb{Z}_n \to \mathbb{Z}_m$ (agreed by prover and verifier), input wire set $A = \{a_0, \ldots, a_{n-1}\}$, output wire set $O = \{o_0, \ldots, o_{m-1}\}$, cleartext input $x \in \mathbb{Z}_n$, global correlation $\Delta \in \mathbb{Z}_p^*$ (fixed, secret, known only to prover), input nonces $R_{A,i} \in \mathbb{Z}_p^*$ (uniformly random).
Ensure: Garbled input $X = W_A(x)$, garbled output $W_O(f(x))$.
Prover (Garbler): Algorithm $\mathsf{Gb}(1^\kappa, C)$
1: Use fixed global correlation $\Delta \in \mathbb{Z}_q^*$ (secret, previously chosen for circuit C).
2: Assume input wire nonces $R_{A,i} \in \mathbb{Z}_q^*$ are uniformly random for each $a_i \in A$.
Prover (Garbler): Algorithm $\mathsf{En}(f_e, x)$
3: Input cleartext $x \in \mathbb{Z}_n$.
4: Compute garbled input $X = W_A(x) = (W_A(x)_i)_{i \in \mathbb{Z}_n}$, where $W_A(x)_i = R_{A,i}\Delta^{\delta_i(x)}$ and $\delta_i(x) = 1$ if $i = x$, else 0.
5: Send $X = W_A(x)$ and cleartext x to verifier.
Verifier: Algorithm $\mathsf{Ev}(C, X)$
6: Use known function $f : \mathbb{Z}_n \to \mathbb{Z}_m$ (from circuit C).
7: Receive $X = W_A(x)$ and cleartext $x \in \mathbb{Z}_n$.
8: Compute garbled output $W_O(f(x)) = \breve{f}(W_A(x))$, where $\breve{f}(W_A(x))_k = \prod_{i \in \mathbb{Z}_n : f(i) = k} W_A(x)_i$ for $k \in \mathbb{Z}_m$.
9: Output $W_O(f(x)) = (\breve{f}(W_A(x))_k)_{k \in \mathbb{Z}_m}$.

Algorithm 10 Ideal Functionality $\mathcal{F}_f$: Garbling and Evaluation of $f : \mathbb{Z}_n \to \mathbb{Z}_m$

Require: Security parameter κ, prime $p \approx 2^\kappa$, circuit C with gate computing $f : \mathbb{Z}_n \to \mathbb{Z}_m$, input wire set $A = \{a_0, \ldots, a_{n-1}\}$ and output wire set $O = \{o_0, \ldots, o_{m-1}\}$.
Ensure: Garbled input $W_A(x)$, garbled output $W_O(f(x))$.

Initialization

1: Choose global correlation $\Delta \in \mathbb{Z}_q^*$ (secret, fixed for circuit C) and nonces $R_{A,i} \in \mathbb{Z}_q^*$ for each $a_i \in A$, known only to trusted party.

On input $x \in \mathbb{Z}_n$ from prover P:

2: Compute garbled input $W_A(x) = (W_A(x)_i)_{i \in \mathbb{Z}_n}$, where $W_A(x)_i = R_{A,i}\Delta^{\delta_i(x)}$ and $\delta_i(x) = 1$ if $i = x$, else 0.

3: Send $W_A(x)$ and cleartext x to verifier V; leak $(W_A(x), x)$ to the simulator.

On request from verifier V to evaluate f:

4: Compute garbled output $W_O(f(x)) = (\breve{f}(W_A(x))_k)_{k \in \mathbb{Z}_m}$, where $\breve{f}(W_A(x))_k = \prod_{i \in \mathbb{Z}_n : f(i) = k} W_A(x)_i$.

5: Send $W_O(f(x))$ to V and leak it to the simulator.

On (Verify, Y') from verifier V:

6: Return 1 if $Y' = W_O(f(x))$ for the x recorded this session, else return 0.

On adaptive corruption of prover P by $\mathcal{A}$:

7: **if** prover P is corrupted **then** Reveal Δ and $\{R_{A,i}\}_{i \in \mathbb{Z}_n}$ to the simulator Sim, which produces the corrupted party's state handed to $\mathcal{A}$ (Algorithm 11 discards these values and reveals its own consistent randomness).

8: **end if**

Lemma 10. *Let $\mathcal{F}_f$ be the ideal functionality of Algorithm 10, augmented with the* Verify *interface (line 6). In any ideal execution with honest prover P and arbitrary verifier $V^\star$, no $V^\star$ can make* Verify *accept a $Y' \neq W_O(f(x))$; hence the ideal-world authenticity advantage is exactly* 0.

Proof. By construction of line 6, Verify returns 1 only on the unique recorded output $W_O(f(x))$; any other string is rejected deterministically. The authenticity experiment of Definition 4, instantiated in the ideal world with Verify as the decode/acceptance check, therefore outputs 1 with probability 0. □

Definition 44. In any vector produced by the protocol, let $t^\star$ denote the unique hot index (that is the only position multiplied by Δ *outside* the call to X). Write $U^\star \in \mathbb{G}$ for the group element fed to X at $t^\star$. For the 2-input tensor product (Theorem 2), we have $U^\star = [R_{A,a}R_{B,b}]\widetilde{G}^{(2)}$ at hot index (a, b). For an I-input tensor product (Remark 22), we have $U^\star = [\prod_j R_{A_j,a_j}]\widetilde{G}^{(I)}$ at hot index $(a_0, \ldots, a_{I-1})$. For `word2hot`, we have $U^\star = [r\Delta^a]G'$ at hot coordinate a.

Lemma 11. *In an honest-prover, corrupt-verifier execution, for each object produced by the prover the corresponding unblinded hot input $U^\star \in \langle G \rangle$ is computationally indistinguishable from a uniform element of $\langle G \rangle$ from the verifier's point of view, under Definition 41. In particular, for at most q_X oracle queries,*

$$\Pr[\textit{the verifier queries } \mathcal{G}_{\mathsf{RO}}^X \textit{ on } U^\star] \leq q_X \cdot \mathsf{Adv}_{S,t}^{\Delta\mathsf{P}}(\kappa) + \frac{nm}{q}.$$

Proof. We analyze each object type from Definition 44 separately, showing that U^* has high min-entropy from the verifier's view.

- **Case 1: 2-input tensor product $T_{A\otimes B}(a, b)$.** By Definition 44, $U^\star = [R_{A,a}R_{B,b}]\widetilde{G}^{(2)}$ where $\widetilde{G}^{(2)} = [k^{(2)}]G$ for $k^{(2)} \xleftarrow{\$} \mathbb{Z}_q^*$ sampled by the prover. The verifier's view consists of:

- Garbled inputs $W_A(a), W_B(b)$ with hot wire values $\chi := R_{A,a}\Delta$ and $\upsilon := R_{B,b}\Delta$;
- Help $H = \Delta \prod_{i,j} X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)})$;
- Per-arity generator powers $\widetilde{G}_0^{(2)} = [k^{(2)}]G$ and $\widetilde{G}_{-1}^{(2)} = [k^{(2)}\Delta^{-1}]G$, but *not* $\widetilde{G}_{-j}^{(2)}$ for $j \geq 2$;
- Per-arity powers $\widetilde{G}_{-j}^{(I')}$ for other arities $I' \neq 2$ used in the circuit, $j \in \{0, \ldots, I'-1\}$;
- Random oracle responses for prior queries.

- **Step 1: Reduction to ΔP-DH.** We build a PPT $\mathcal{B}$ that, given a $(\{0,-1\},-2)$-ΔP-DH instance $(G,\ A_0 := [k\Delta^0]G,\ A_{-1} := [k\Delta^{-1}]G)$, uses any $V^\star$ that queries $U^\star$ with non-negligible probability to output $[k\Delta^{-2}]G$. $\mathcal{B}$ sets $\widetilde{G}_0^{(2)} := A_0$ and $\widetilde{G}_{-1}^{(2)} := A_{-1}$ (implicitly fixing $k^{(2)} = k$ and the correlation Δ, both unknown to $\mathcal{B}$), and simulates the view:
  - Sample the cold nonces $R_{A,i}$ $(i \neq a)$, $R_{B,j}$ $(j \neq b) \xleftarrow{\$} \mathbb{Z}_q^*$, and sample the hot wire values $\chi := W_A(a)_a,\ \upsilon := W_B(b)_b \xleftarrow{\$} \mathbb{Z}_q^*$ directly (in the real world $\chi = R_{A,a}\Delta$, $\upsilon = R_{B,b}\Delta$ are uniform; sampling them fixes $R_{A,a}, R_{B,b}$ implicitly). Send $W_A(a), W_B(b), (a,b)$ to $V^\star$.
  - Cold cell (i,j), $i \neq a, j \neq b$: compute the group element $[R_{A,i}R_{B,j}]\widetilde{G}_0^{(2)}$ and answer/program the oracle there with a fresh $z_{ij} \xleftarrow{\$} \mathbb{Z}_q^*$.
  - Mixed cell (say $i = a, j \neq b$): the wire product is $\chi R_{B,j}$; compute $[\chi R_{B,j}]\widetilde{G}_{-1}^{(2)}$ and program it with a fresh z_{ij}. (Symmetrically for $i \neq a, j = b$.)
  - Sample the hot output $T^\star \xleftarrow{\$} \mathbb{Z}_q^*$ (in the real world $T^\star = \Delta\, X(U^\star)$, uniform while $U^\star$ is unqueried), and set the help $H := T^\star \cdot \prod_{(i,j)\neq(a,b)} z_{ij}$, so that the verifier's reconstruction $H \cdot h^{-1}$ returns $T^\star$.

  Every step uses only A_0, A_{-1} and scalars $\mathcal{B}$ sampled; $\mathcal{B}$ never needs k, Δ, or $U^\star$. The view is identically distributed to the real one up to the event that $V^\star$ queries $U^\star$.
- **Step 2: Extraction.** $\mathcal{B}$ watches $V^\star$'s oracle queries. For each query Q, using the known scalar $\sigma := \chi\upsilon = R_{A,a}R_{B,b}\Delta^2$ it tests

$$Q \stackrel{?}{=} [\sigma]\,\widetilde{G}_{-2}^{(2)} \quad \text{i.e. whether } [\sigma^{-1}]Q = \widetilde{G}_{-2}^{(2)} = [k\Delta^{-2}]G.$$

  Concretely $\mathcal{B}$ cannot test against $\widetilde{G}_{-2}^{(2)}$ directly (that is the unknown), nor can it recognise which of $V^\star$'s queries, if any, equals $U^\star$. It therefore samples an index $\iota \xleftarrow{\$} \{1, \ldots, q_X\}$ at the outset, lets Q_ι denote the ι-th oracle query of $V^\star$ falling outside the set of cold/mixed points it programmed, and outputs $\widehat{W} := [\sigma^{-1}]Q_\iota$. If $V^\star$ queries $U^\star = [R_{A,a}R_{B,b}]\widetilde{G}_0^{(2)} = [\sigma]\widetilde{G}_{-2}^{(2)}$ at any point, then with probability at least $1/q_X$ the guess satisfies $Q_\iota = U^\star$, in which case $\widehat{W} = \widetilde{G}_{-2}^{(2)} = [k\Delta^{-2}]G$ is the ΔP-DH solution. Hence $\mathsf{Adv}^{\Delta\mathsf{P}}_{\{0,-1\},-2}(\kappa) \geq \frac{1}{q_X}\big(\Pr[\mathsf{HotQuery}] - \frac{nm}{q}\big)$, the subtracted term accounting for an accidental collision of $U^\star$ with a programmed point.
- **Step 3: Bound.** $V^\star$'s view is independent of whether the hot output was sampled ($T^\star$) or real ($\Delta X(U^\star)$) unless $U^\star$ is queried; each such query lets $\mathcal{B}$ win ΔP-DH (up to the chance that an unrelated programmed point collides, $\leq nm/q$). Hence over at most q_X queries,

$$\Pr[\mathsf{HotQuery}] \leq q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}_{\{0,-1\},-2}(\kappa) + \frac{nm}{q},$$

which is negligible. This establishes the lemma for the 2-input tensor case; Case I is identical with $(\{0,\ldots,-(I-1)\},-I)$-ΔP-DH, and `word2hot` (Case 2) with the missing-middle instance.

- **Case 2: word2hot conversion (Section 4.4.4).** By Definition 44, $U^\star = [r\Delta^a]G'$ where $G' = [K]G$ for a uniformly random $K \in \mathbb{Z}_q^*$ chosen by the prover. The verifier receives:
  - Word garbling $\Omega(a) = r\Delta^a$;
  - Cleartext $a \in \mathbb{Z}_n$;
  - Generators $G'_i = [\Delta^i]G'$ for $i \in \{-(n-1),\ldots,n-1\} \setminus \{0\}$.

  We observe that $G'_0 = G'$ is *not* provided to the verifier. The verifier can compute

  $$[\Omega(a)]G'_c = [r\Delta^{a+c}]G' \quad \text{for } c \neq 0,$$

  but cannot directly compute $[\Omega(a)]G' = [r\Delta^a]G' = U^*$ without access to G'.

  Now, $U^\star = [\Omega(a)]G'_0$ with $\Omega(a)$ known to the verifier, so producing $U^\star$ is equivalent to computing the withheld middle power $G'_0 = [K]G$ from $\{[K\Delta^i]G\}_{i\in\{\pm1,\ldots,\pm(n-1)\}}$: exactly the (S,t)-ΔP-DH instance with $S = \{\pm1,\ldots,\pm(n-1)\}$, $t = 0$ (Definition 41). The reduction is Steps 1-3 of Case 1 verbatim, planting the challenge powers as the G'_i and applying the same index-guessing extraction: the guessed query Q_ι, corrected by the known scalar as $[\Omega(a)^{-1}]Q_\iota$, is the missing power. Hence $\Pr[\mathsf{HotQuery}] \leq q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}_{S,0}(\kappa) + n/q$.

- **Case 3: I-input tensor product, $I \geq 3$.** By Definition 44, $U^\star = [\prod_j R_{A_j,a_j}]\widetilde{G}^{(I)}_0$. After the material-list correction the verifier holds $\widetilde{G}^{(I)}_{-j}$ for $j \in \{0,\ldots,I-1\}$ but not $\widetilde{G}^{(I)}_{-j}$ for $j \geq I$; the fully hot product carries Δ^I, so producing $U^\star$ amounts to computing the withheld power $\widetilde{G}^{(I)}_{-I}$. The argument is Steps 1-3 of Case 1 with the $(\{0,-1,\ldots,-(I-1)\},-I)$-ΔP-DH instance: the challenge powers are planted as $\widetilde{G}^{(I)}_0,\ldots,\widetilde{G}^{(I)}_{-(I-1)}$, and a hot query, corrected by the known hot wire product, yields the missing power. Detection of a hot query proceeds via the analogous test $Q \stackrel{?}{=} [\prod_j R_{A_j,a_j}]P$. Cross-arity collisions are bounded between arity I and any other arity $I' \neq I$ used in the circuit, by independence of $k^{(I)}$ and $k^{(I')}$, with the same bound as Step 3 of Case 1.

Combining all cases, $\Pr[\mathsf{HotQuery}] \leq q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}_{S,t}(\kappa) + \mathrm{poly}(\kappa)/q$, as claimed (the polynomial being nm, n, or $\prod_j n_j$ per case). □

Lemma 12. *Consider an honest-prover, corrupt-verifier execution of any OHMG object containing hot oracle entries (Definition 44) in the GUC-RO model with global $\mathcal{G}^X_{\mathrm{RO}}$. Let q_X be the number of X-oracle queries issued by $V^\star$ during the session, $q_X^{\mathcal{Z}}$ the number issued by $\mathcal{Z}$ before the prover samples its randomness, $|U|$ the number of non-hot entries, and q the group order. Then the view of $V^\star$ with $\Delta \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$ is computationally indistinguishable from its view with an independent $\Delta' \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$, with distinguishing advantage at most $q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}_{S,t}(\kappa) + \mathrm{poly}(\kappa)/q + |U| \cdot q_X^{\mathcal{Z}}/q$, where the polynomial collects the programmed-point collision terms of Lemma 11.*

Proof. All scalars are in $\mathbb{Z}_q^*$ and X is realized by the global random oracle $\mathcal{G}^X_{\mathrm{RO}}$ with session/domain separation. Write $U_t = [\xi_t]B_t$ for the group elements the prover feeds to X while computing tables/vectors, and let $U^\star$ be, for each such object, the input at its hot index (Definition 44). Let h denote the product of all non-hot X-outputs in a given object, and recall $H = \Delta \cdot \prod_t X(U_t)$ from equation (4).

- G_0 **(real)**: Run the real execution with $\Delta \xleftarrow{\$} \mathbb{Z}_q^*$.

- G_1 (lazy RO): Answer all X-queries by lazy sampling: upon first query on $U \in \mathbb{G}$, sample $X(U) \xleftarrow{\$} \mathbb{Z}_q^*$ and memorize. Identical to G_0.

- G_2 **(fix all non-hot entries)**: The hybrid replaces, for each table/vector and each non-hot index $t \neq t^\star$, the oracle answer $X(U_t)$ by a fresh independent value. This is a game-hop internal to the analysis, not an operation available to any protocol party: since $\mathcal{G}_{\mathrm{RO}}^X$ answers by lazy sampling (hybrid G_1), reassigning the value at a point that no party has yet queried induces an identical distribution. The event that some point U_t was already queried is accounted for below. Let h be their product and set the hot output to $H \cdot h^{-1}$. By Lemma 7, this equals $\Delta \cdot X(U^\star)$ and preserves the joint distribution of all entries. Let $\mathsf{HotQuery} = \bigvee V^*$ queries X on an unblinded hot input $U^\star$. By Lemma 11, applied to each object containing a hot entry, $\Pr[\mathsf{HotQuery}] \leq q_X \cdot \mathsf{Adv}_{S,t}^{\Delta\mathsf{P}}(\kappa) + \mathrm{poly}(\kappa)/q$, where the polynomial collects the programmed-point collision terms of all objects. Hence $\left|\Pr[G_1 = 1] - \Pr[G_2 = 1]\right| \leq \Pr[\mathsf{HotQuery}]$.

  The programming succeeds with overwhelming probability: the cold inputs U_t for $t \neq t^\star$ are unpredictable to the environment $\mathcal{Z}$ before the prover samples its randomness, since these inputs depend on fresh uniform nonces. The probability that $\mathcal{Z}$ has queried any of these points beforehand is at most $|U| \cdot q_X^{\mathcal{Z}}/q = \mathsf{negl}(\kappa)$, where $|U|$ is the total number of non-hot entries across all tables/vectors.

- G_3 **(switch Δ)**: Define G_3 to be the same as G_2, except that we sample an independent $\Delta' \xleftarrow{\$} \mathbb{Z}_q^*$ instead of Δ and use Δ' in the computation of all help values and hot entries. Conditioned on $\neg\mathsf{HotQuery}$, the verifier never queries X on an unblinded hot input $U^\star$. Therefore all X-outputs that appear in its view are either:

  – values $X(U)$ for inputs U that do not involve Δ at all, or

  – (for the hot entries) products of the form $\Delta \cdot Z_{ab}$ where Z_{ab} is an X output on a point $U^\star$ that was never queried.

  In the second case, Z_{ab} is uniform in $\mathbb{Z}_q^*$ and unknown to the verifier, hence $\Delta \cdot Z_{ab}$ is uniform in $\mathbb{Z}_q^*$ and its distribution is independent of the concrete value of Δ. Replacing Δ by Δ' therefore leaves the joint distribution of the verifier's view unchanged: conditioned on $\neg\mathsf{HotQuery}$ we have $\Pr[G_2 = 1 \mid \neg\mathsf{HotQuery}] = \Pr[G_3 = 1 \mid \neg\mathsf{HotQuery}]$, and hence $|\Pr[G_2 = 1] - \Pr[G_3 = 1]| = 0$.

Summing the losses, we get $\left|\Pr[G_0 = 1] - \Pr[G_3 = 1]\right| \leq \Pr[\mathsf{HotQuery}] + |U| \cdot q_X^{\mathcal{Z}}/q \leq q_X \cdot \mathsf{Adv}_{S,t}^{\Delta\mathsf{P}}(\kappa) + \mathrm{poly}(\kappa)/q + |U| \cdot q_X^{\mathcal{Z}}/q$. □

Algorithm 11 Ideal-World Simulator Sim_f, honest prover

Require: Gate $f : \mathbb{Z}_n \to \mathbb{Z}_m$, ideal functionality $\mathcal{F}_f$, adversary $\mathcal{A}$ (corrupt verifier). Standing: κ, p.

Ensure: View indistinguishable from real-world π_f.

Setup and garbling.

1: On input x from $\mathcal{Z}$, forward to $\mathcal{F}_f$ and receive the leak $(W_A(x), x, W_O(f(x)))$.

2: Send $(W_A(x), x)$ to $\mathcal{A}$.

Output.

3: On receipt of $W_O(f(x))$ from $\mathcal{F}_f$, forward to $\mathcal{A}$.

Adaptive corruption.

4: If $\mathcal{A}$ corrupts the prover, reveal $(\Delta', \{R'_{A,i}\})$, the randomness used to build the transcript, to $\mathcal{A}$; consistency is immediate. The values leaked by $\mathcal{F}_f$ (line 7) are discarded.

Theorem 4. *Let π_f be the protocol for garbling and evaluating the function $f : \mathbb{Z}_n \to \mathbb{Z}_m$ as defined in Algorithm 9, and let $\mathcal{F}_f$ be the corresponding ideal functionality as defined in Algorithm 10. For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ controlling a corrupt verifier and any environment $\mathcal{Z}$, the real-world execution $\mathsf{REAL}_{\mathcal{A},\pi_f}(\mathcal{Z})$ is computationally indistinguishable from the ideal-world execution $\mathsf{IDEAL}_{\mathsf{Sim}_f,\mathcal{F}_f}(\mathcal{Z})$ with simulator Sim_f as defined in Algorithm 11. The advantage $\mathsf{Adv}_{\mathcal{Z}}(\kappa)$ is negligible (that is, $\mathsf{Adv}_{\mathcal{Z}}(\kappa) \le \mathsf{negl}(\kappa)$) for security parameter κ, ensuring that π_f securely realizes $\mathcal{F}_f^{-\mathsf{Verify}}$ (that is, $\mathcal{F}_f$ without its* Verify *clause) with UC authenticity in the honest prover, corrupt verifier scenario with adaptive prover corruption.*

Proof. Let $\mathcal{A}$ be a PPT adversary controlling the corrupt verifier $V^\star$, and $\mathcal{Z}$ a PPT environment. We show that $\mathsf{REAL}_{\mathcal{A},\pi_f}(\mathcal{Z}) \equiv \mathsf{IDEAL}_{\mathsf{Sim}_f,\mathcal{F}_f^{-\mathsf{Verify}}}(\mathcal{Z})$ with advantage exactly 0, where $\mathcal{F}_f^{-\mathsf{Verify}}$ denotes $\mathcal{F}_f$ with the Verify clause removed; the Verify interface of $\mathcal{F}_f$ is an ideal-world analysis device used only in Lemma 10 and Corollary 3 (its real-world counterpart is the hash check against the output-wire commitments that Gb posts to $\mathcal{G}_{\mathsf{trust}}$ when the gate is deployed in a garbled circuit, i.e. in the one-object instance of Theorem 6; the stand-alone π_f posts nothing). π_f issues no $\mathcal{G}_{\mathsf{RO}}^X$ query, so no oracle term arises.

Since π_f makes no $\mathcal{G}_{\mathsf{RO}}^X$ query and posts nothing, the transcript is the deterministic image $(W_A(x), x)$ of the prover randomness $(\Delta, \{R_{A,i}\})$, which appears only as fresh uniform scalars; the simulator's resampled $(\Delta', \{R'_{A,i}\})$ therefore induces an identically distributed view. The delivered output is computed by $\mathcal{F}_f$ with the same evaluation rule as $\breve{f}$ (Algorithm 10), so the real and ideal executions coincide: $\mathsf{Adv}_{\mathcal{Z}}(\kappa) = 0$, proving UC realization.

Concerning adaptive corruption, if $\mathcal{Z}$ corrupts the prover at any point, Sim_f outputs the randomness $(\Delta', \{R'_{A,i}\})$ used in H_2. Consistency with the transcript follows by construction: the only message sent prior to corruption is $W'_A(x)$ (line 2), which equals $R'_{A,i}\Delta'^{\delta_i(x)}$ by definition. The protocol π_f does not invoke $\mathcal{G}_{\mathsf{RO}}^X$, so no random-oracle programming is needed and the corruption response is information-theoretic.

Authenticity follows from the UC realization together with Lemma 10 via the standard reduction (Corollary 3). □

Corollary 3. *In the honest-prover, malicious-verifier setting, protocol π_f satisfies authenticity in the sense of Definition 4.*

Proof. By Lemma 10, any ideal execution with $\mathcal{F}_f$ enjoys perfect authenticity. If a PPT adversary $\mathcal{A}$ against π_f could win the authenticity game with non-negligible probability, we could build an environment $\mathcal{Z}$ that simply runs that game inside the UC experiment and outputs 1 if, and only if, $\mathcal{A}$ produces a successful forgery.

By Theorem 4 (and Definition 39), the outputs of $\mathcal{Z}$ in the real and ideal executions must be computationally indistinguishable. But in the ideal execution the forgery probability is 0, whereas in the real execution it is non-negligible by assumption. This contradicts UC security, hence the forging advantage against π_f must be negligible.

In the real execution, $\mathcal{Z}$ adjudicates forgery success by comparing Hash of the candidate output labels against the output-wire commitments that Gb posts to $\mathcal{G}_{\mathsf{trust}}$ when the gate is deployed in a garbled circuit, the one-object instance of Theorem 6. The stand-alone protocol posts no commitments (Remark 38), which is precisely what permits its adaptive-corruption claim; since no corruption occurs inside the authenticity experiment itself, the committed adjudication and the adaptive claim never apply to the same execution. The comparison is readable by $\mathcal{Z}$ because $\mathcal{G}_{\mathsf{trust}}$ is global, and realizes the acceptance check of Definition 4 without access to the prover's secrets. □

5.2.1 Tensor products and several functions maps

In this section we prove authenticity of the tensor product evaluation, and from it, the security of the evaluation of a function, and UC security, it follows the authenticity of evaluation of functions of several variables.

Proposition 2. *The outcome*

$$T_{A\otimes B}(a,b)_{ij} = X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)})\Delta^{\delta_i(a)\delta_j(b)}$$

is, at the hot index $(i,j)=(a,b)$, uniformly distributed on $\mathbb{Z}_q^*$ over the garbler's coins, and remains computationally indistinguishable from uniform for any PPT adversary whose view excludes the help H (equivalently: before H is delivered), given that $R_{A,i}$, $R_{B,j}$ are uniform and Δ is a fixed secret value. Once H is delivered the hot entry is determined by the view (it equals Hh^{-1}); the conditional unpredictability statement used by the security proofs is the one of Lemma 11, which bounds the probability that the adversary queries the underlying oracle input $U^\star$.

Remark 35. We stress that, since OHMG is privacy-free, the position (a,b) of the hot cell is public and the evaluator can recompute every cold entry from her own material; Proposition 2 therefore asserts unpredictability only at the hot index, and not that the joint distribution hides which cell is hot.

Proof. In the GUC-RO hybrid model (Definition 8): the product $R_{A,i}R_{B,j}$ of independent uniforms is uniform on $\mathbb{Z}_q^*$, so $[R_{A,i}R_{B,j}]\widetilde{G}^{(2)}$ is uniform on $\langle G\rangle\setminus\{\mathcal{O}\}$; the adversary has queried this point with probability at most $q_X/(q-1)$, so X at it is a fresh uniform value; and multiplication by the nonzero factor $\Delta^{\delta_i(a)\delta_j(b)}\in\{1,\Delta\}$ permutes $\mathbb{Z}_q^*$. The claim follows. □

Algorithm 12 Protocol π_{TP}: Verifier's Evaluation of Tensor Product

Require: Security parameter κ, prime $p \approx 2^\kappa$, circuit C with gate computing the tensor product $T_{A\otimes B}$ of one-hot inputs of sizes n and m, input wire sets $A = \{a_0, \ldots, a_{n-1}\}$ and $B = \{b_0, \ldots, b_{m-1}\}$, output wire set $O = \{o_{i,j} \mid i \in \mathbb{Z}_n, j \in \mathbb{Z}_m\}$, garbled inputs $W_A(a) = (W_A(a)_i)_{i\in\mathbb{Z}_n}$, $W_B(b) = (W_B(b)_j)_{j\in\mathbb{Z}_m}$, cleartext inputs $a \in \mathbb{Z}_n$, $b \in \mathbb{Z}_m$, generators $\widetilde{G}^{(2)}, \widetilde{G}^{(2)}_{-1} \in \langle G\rangle$, function $X : \langle G\rangle \to \mathbb{Z}_q^*$, help value $H \in \mathbb{Z}_q^*$ from prover.
Ensure: Output $T_{A\otimes B}(a, b)$, encoding the one-hot tensor product.

Initialize empty list $T_{A\otimes B} = []$
for $i \in \mathbb{Z}_n$ **do**
  for $j \in \mathbb{Z}_m$ **do**
    if $i \neq a$ **and** $j \neq b$ **then** $w_{i,j} \leftarrow X([W_A(a)_i \cdot W_B(b)_j]\widetilde{G}^{(2)})$
    else if $i \neq a$ **xor** $j \neq b$ **then** $w_{i,j} \leftarrow X([W_A(a)_i \cdot W_B(b)_j]\widetilde{G}^{(2)}_{-1})$
    else $w_{a,b} \leftarrow 1$ ▷ $i = a$ **and** $j = b$
    end if
    Append $w_{i,j} = T_{A\otimes B}(a, b)_{ij}$ to $T_{A\otimes B}$
  end for
end for
Compute $h \leftarrow \prod_{i\in\mathbb{Z}_n, j\in\mathbb{Z}_m} w_{i,j}$
Compute $T_{A\otimes B}(a, b)_{a,b} \leftarrow H \cdot h^{-1}$ in $\mathbb{Z}_q^*$ ▷ Update hot wire using help H
return $T_{A\otimes B}(a, b)$ ▷ Return full tensor product output

Algorithm 13 Ideal Functionality $\mathcal{F}_{\mathsf{TP}}$: Evaluation of Tensor Product $T_{A\otimes B}$

Require: Security parameter κ, prime $p \approx 2^\kappa$, circuit C with gate computing the tensor product $T_{A\otimes B}$ of one-hot inputs of sizes n and m, input wire sets $A = \{a_0, \ldots, a_{n-1}\}$ and $B = \{b_0, \ldots, b_{m-1}\}$, output wire set $O = \{o_{i,j} \mid i \in \mathbb{Z}_n, j \in \mathbb{Z}_m\}$.
Ensure: Garbled inputs $W_A(a)$, $W_B(b)$, output $T_{A\otimes B}(a, b)$.

Choose $\Delta \xleftarrow{\$} \mathbb{Z}_q^*$, nonces $R_{A,i}, R_{B,j} \xleftarrow{\$} \mathbb{Z}_q^*$ for each $a_i \in A$, $b_j \in B$, and the per-arity secret $k^{(2)} \xleftarrow{\$} \mathbb{Z}_q^*$ with $\widetilde{G}^{(2)} := [k^{(2)}]G$.
Compute help $H = \Delta \prod_{i\in\mathbb{Z}_n, j\in\mathbb{Z}_m} X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)})$ (Definition 25).
On input $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_m$ **from prover** P**:**
Compute $W_A(a)_i = R_{A,i}\Delta^{\delta_i(a)}$ and $W_B(b)_j = R_{B,j}\Delta^{\delta_j(b)}$. ▷ $\delta_i(-)$ is Kronecker's delta
Send $W_A(a)$, $W_B(b)$, cleartext (a, b), and H to verifier V.
On request from verifier V **to evaluate** $T_{A\otimes B}$**:**
Compute $T_{A\otimes B}(a, b)_{i,j} = X([W_A(a)_i \cdot W_B(b)_j]\widetilde{G}^{(2)})$ for $i \neq a$ and $j \neq b$.
Compute $T_{A\otimes B}(a, b)_{i,j} = X([W_A(a)_i \cdot W_B(b)_j]\widetilde{G}^{(2)}_{-1})$ for $i \neq a$ xor $j \neq b$.
Set $T_{A\otimes B}(a, b)_{a,b} = 1$ and compute $h = \prod_{i\in\mathbb{Z}_n, j\in\mathbb{Z}_m} T_{A\otimes B}(a, b)_{i,j}$.
Update $T_{A\otimes B}(a, b)_{a,b} \leftarrow H \cdot h^{-1}$ in $\mathbb{Z}_q^*$ and send $T_{A\otimes B}(a, b)$ to V.
On (Verify, T') **from verifier** V**:**
Return 1 if $T' = T_{A\otimes B}(a, b)$ for the input (a, b) recorded this session, else return 0.
if prover P is corrupted **then** Reveal Δ, $\{R_{A,i}\}_{i\in\mathbb{Z}_n}$, $\{R_{B,j}\}_{j\in\mathbb{Z}_m}$, and $k^{(2)}$ to the simulator Sim, which produces the corrupted party's state handed to $\mathcal{A}$ (Algorithm 14 discards these values and reveals its own consistent randomness).
end if

Algorithm 14 Ideal-World Simulator $\mathsf{Sim}_{\mathsf{TP}}$, honest prover

Require: Tensor gate computing $T_{A\otimes B}$ of one-hot inputs of sizes n and m, ideal functionality $\mathcal{F}_{\mathsf{TP}}$, adversary $\mathcal{A}$ (corrupt verifier). Standing: κ, p, G, Δ'.

Ensure: View indistinguishable from real-world π_{TP}.

Setup.

On input (a, b) from $\mathcal{Z}$, forward to $\mathcal{F}_{\mathsf{TP}}$. Sample $\Delta', \{R'_{A,i}\}_i, \{R'_{B,j}\}_j \xleftarrow{\$} \mathbb{Z}_q^*$ (per-session) and $k^{(2)\prime} \xleftarrow{\$} \mathbb{Z}_q^*$ (per-circuit, sampled once and reused for all arity-2 tensor gates in C); compute $\widetilde{G}^{(2)\prime} := [k^{(2)\prime}]G$ and $\widetilde{G}^{(2)\prime}_{-1} := [\Delta'^{-1}]\widetilde{G}^{(2)\prime}$.

Simulate garbling.

Compute $W'_A(a)_i := R'_{A,i}\Delta'^{\delta_i(a)}$ and $W'_B(b)_j := R'_{B,j}\Delta'^{\delta_j(b)}$ as in π_{TP}.

Receive $T_{A\otimes B}(a, b)$ from $\mathcal{F}_{\mathsf{TP}}$.

For each cold $(i, j) \neq (a, b)$: sample $z'_{ij} \xleftarrow{\$} \mathbb{Z}_q^*$ and call $\mathsf{Program}(\mathsf{sid}, \mathsf{tag}_{\mathsf{TP}}, [R'_{A,i}R'_{B,j}]\widetilde{G}^{(2)\prime}, z'_{ij})$ on $\mathcal{G}^X_{\mathsf{RO}}$.

Set $z'_{a,b} := \Delta'^{-1} \cdot T_{A\otimes B}(a, b)_{a,b}$ and call $\mathsf{Program}(\mathsf{sid}, \mathsf{tag}_{\mathsf{TP}}, [R'_{A,a}R'_{B,b}]\widetilde{G}^{(2)\prime}, z'_{a,b})$ on $\mathcal{G}^X_{\mathsf{RO}}$.

Set $H' := \Delta' \cdot \prod_{i,j} z'_{ij}$.

Send $\big(W'_A(a), W'_B(b), (a, b), \widetilde{G}^{(2)\prime}_{-1}, H'\big)$ to $\mathcal{A}$.

Assemble T' with $T'_{i,j} := z'_{ij}$ for $(i, j) \neq (a, b)$ and $T'_{a,b} := \Delta' \cdot z'_{a,b} = T_{A\otimes B}(a, b)_{a,b}$; forward T' to $\mathcal{A}$.

Adaptive corruption.

If $\mathcal{A}$ corrupts the prover, reveal $(\Delta', \{R'_{A,i}\}, \{R'_{B,j}\}, k^{(2)\prime})$ to $\mathcal{A}$, consistent with the transcript.

Lemma 13. *Let $\mathcal{F}_{\mathsf{TP}}$ be the ideal functionality of Algorithm 13. In any ideal execution with honest prover P and arbitrary (possibly malicious) verifier $V^\star$, the adversary's success probability in the authenticity experiment of Definition 4, applied to the tensor product output $T_{A\otimes B}(a, b)$, is* 0.

Proof. Exactly as in Lemma 10: in the ideal world $V^\star$ only receives the garbled input $(W_A(a), W_B(b))$ for the actual input (a, b) and a single corresponding tensor-product output $T_{A\otimes B}(a, b)$, and never any labels for different inputs $(a', b') \neq (a, b)$ or different outputs. Thus any forged output that decodes to a valid value for $(a', b') \neq (a, b)$ is impossible, and the authenticity game outputs 1 with probability 0. □

Theorem 5. *The protocol π_{TP} (Algorithm 12) securely realizes $\mathcal{F}_{\mathsf{TP}}^{-\mathsf{Verify}}$ (Algorithm 13 without its* Verify *clause) with simulator $\mathsf{Sim}_{\mathsf{TP}}$ (Algorithm 14), ensuring UC authenticity in the honest prover, corrupt verifier scenario with adaptive prover corruption.*

Proof. Let $\mathcal{A}$ be a PPT adversary controlling the corrupt verifier $V^\star$, and $\mathcal{Z}$ a PPT environment. We show that $\mathsf{REAL}_{\mathcal{A},\pi_{\mathsf{TP}}}(\mathcal{Z}) \approx_c \mathsf{IDEAL}_{\mathsf{Sim}_{\mathsf{TP}},\mathcal{F}_{\mathsf{TP}}^{-\mathsf{Verify}}}(\mathcal{Z})$ (with $\mathcal{F}_{\mathsf{TP}}^{-\mathsf{Verify}}$ denoting $\mathcal{F}_{\mathsf{TP}}$ with the Verify clause removed, as in Theorem 4; Verify is the ideal-world analysis device of Lemma 13 and Corollary 5) with advantage at most $q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) + (|U| + nm) \cdot q_X^{\mathcal{Z}}/q + \big((nm)^2 + nm\big)/q$, where q_X is the total number of $\mathcal{G}^X_{\mathsf{RO}}$ queries made by $V^\star$ in the session. Note that $q_X = \mathsf{poly}(\kappa)$ for any PPT adversary, so $q_X/q = \mathsf{negl}(\kappa)$.

We use a sequence of three hybrids H_0, H_1, H_2.

- H_0 **(Real-world execution).** $\mathcal{A}$ interacts with the honest prover running π_{TP} on input (a, b). The prover samples $\Delta \xleftarrow{\$} \mathbb{Z}_q^*$, $\{R_{A,i}\}_{i\in\mathbb{Z}_n} \xleftarrow{\$} (\mathbb{Z}_q^*)^n$, and $\{R_{B,j}\}_{j\in\mathbb{Z}_m} \xleftarrow{\$} (\mathbb{Z}_q^*)^m$, samples the per-arity secret $k^{(2)} \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $\widetilde{G}^{(2)} = [k^{(2)}]G$, $\widetilde{G}^{(2)}_{-1} = [\Delta^{-1}]\widetilde{G}^{(2)}$ (Section 4.3.1). The prover then computes the garbled inputs $W_A(a)_i =$

$R_{A,i}\Delta^{\delta_i(a)}$, $W_B(b)_j = R_{B,j}\Delta^{\delta_j(b)}$, and the help $H = \Delta\prod_{i,j} X([R_{A,i}R_{B,j}]\widetilde{G}^{(2)})$, sending $(W_A(a), W_B(b), \widetilde{G}^{(2)}_{-1}, H, (a,b))$ to $\mathcal{A}$. The verifier evaluates $T_{A\otimes B}(a,b)$ via Definition 27 and Theorem 2.

- H_1 **(Independent prover randomness).** Identical to H_0, except that the prover's randomness $(\Delta, \{R_{A,i}\}, \{R_{B,j}\}, k^{(2)})$ is replaced by an independent fresh sample

$$(\Delta', \{R'_{A,i}\}, \{R'_{B,j}\}, k^{(2)\prime})$$

from the same distribution. The garbled inputs and help are recomputed from these primed values:

$$W'_A(a)_i = R'_{A,i}\Delta'^{\delta_i(a)}, \quad W'_B(b)_j = R'_{B,j}\Delta'^{\delta_j(b)},$$

$$H' = \Delta'\prod_{i,j} X([R'_{A,i}R'_{B,j}]\widetilde{G}^{(2)\prime}),$$

where $\widetilde{G}^{(2)\prime} = [k^{(2)\prime}]G$ and $\widetilde{G}^{(2)\prime}_{-1} = [\Delta'^{-1}]\widetilde{G}^{(2)\prime}$.

- H_2 **(Ideal-world execution).** Identical to H_1 in setup, but the output $T_{A\otimes B}(a,b)$ is delivered by the ideal functionality $\mathcal{F}_{\mathsf{TP}}$ to $V^\star$ (via the simulator) instead of being computed locally by $V^\star$ from the help H'.

We now discuss the indistinguishability between hybrids:

- **Indistinguishability $H_0 \approx_s H_1$.** By Lemma 12, the verifier's view in H_0 (with Δ) is computationally indistinguishable from its view with an independent Δ', with advantage at most $q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) + |U| \cdot q_X^{\mathcal{Z}}/q + \big((nm)^2 + nm\big)/q$ (Definition 42 upper-bounds the per-family advantage $\mathsf{Adv}^{\Delta\mathsf{P}}_{S,t}$; the $(nm)^2$ term accounts for internal collisions among the gate's programmed points). The lemma applies in the GUC-RO model with global $\mathcal{G}^X_{\mathsf{RO}}$ and addresses precisely the $\Delta \to \Delta'$ replacement. The nonces $\{R_{A,i}\}, \{R_{B,j}\}$ and the per-arity secret $k^{(2)}$ appear only as fresh uniform scalars in both H_0 and H_1; resampling them does not change the joint distribution of the view. Hence $\big|\Pr[H_0 = 1] - \Pr[H_1 = 1]\big| \le q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) + |U| \cdot q_X^{\mathcal{Z}}/q + \big((nm)^2 + nm\big)/q$.

- **Indistinguishability $H_1 \equiv H_2$.** The output $T_{A\otimes B}(a,b)$ is a deterministic function of the prover's randomness $(\Delta', \{R'_{A,i}\}, \{R'_{B,j}\}, k^{(2)\prime})$ and the input (a,b), by Theorem 2. In H_1 the verifier reconstructs this value from H' via the protocol equations; in H_2 the same value is delivered by $\mathcal{F}_{\mathsf{TP}}$. By inspection of Algorithms 12 and 13, both compute $T_{A\otimes B}(a,b)$ via the same expressions, so the two views coincide except when a $\mathsf{Program}$ call fails: $\big|\Pr[H_1 = 1] - \Pr[H_2 = 1]\big| \le nm \cdot q_X^{\mathcal{Z}}/q$.

By the triangle inequality, $\big|\Pr[H_0 = 1] - \Pr[H_2 = 1]\big| \le q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) + (|U| + nm) \cdot q_X^{\mathcal{Z}}/q + \big((nm)^2 + nm\big)/q$, which is negligible in κ. Hence: $\mathsf{REAL}_{\mathcal{A},\pi_{\mathsf{TP}}}(\mathcal{Z}) \approx_c \mathsf{IDEAL}_{\mathsf{Sim}_{\mathsf{TP}},\mathcal{F}_{\mathsf{TP}}}(\mathcal{Z})$, proving UC realization.

Concerning adaptive corruption, if $\mathcal{Z}$ corrupts the prover after the protocol messages have been sent, $\mathsf{Sim}_{\mathsf{TP}}$ reveals $(\Delta', \{R'_{A,i}\}, \{R'_{B,j}\}, k^{(2)\prime})$ to $\mathcal{A}$. We argue this is consistent with the transcript and the global random oracle $\mathcal{G}^X_{\mathsf{RO}}$:

- **Wire label consistency.** The inputs $W'_A(a)_i = R'_{A,i}\Delta'^{\delta_i(a)}$ and $W'_B(b)_j = R'_{B,j}\Delta'^{\delta_j(b)}$ are deterministic functions of the revealed values, matching the messages already sent to $\mathcal{A}$.

- **Per-arity generator consistency.** $\widetilde{G}^{(2)\prime} = [k^{(2)\prime}]G$ and $\widetilde{G}^{(2)\prime}_{-1} = [\Delta'^{-1}]\widetilde{G}^{(2)\prime}$ are deterministic functions of $(k^{(2)\prime}, \Delta')$, matching the value sent in line 7.

- **Help and RO consistency.** $\mathsf{Sim_{TP}}$ programmed $\mathcal{G}_{\mathsf{RO}}^{X}$ at $[R'_{A,i}R'_{B,j}]\widetilde{G}^{(2)\prime}$ to output $z'_{i,j}$, and computed $H' = \Delta' \prod_{i,j} z'_{i,j}$. Post-corruption, $\mathcal{A}$ may query $\mathcal{G}_{\mathsf{RO}}^{X}$ at any $[R'_{A,i}R'_{B,j}]\widetilde{G}^{(2)\prime}$ and obtain $z'_{i,j}$; the product $\Delta' \prod_{i,j} z'_{i,j}$ equals H' by construction. By the unpredictability argument of Lemma 11, all programming calls succeeded except with probability $nm \cdot q_X^{\mathcal{Z}}/q = \mathsf{negl}(\kappa)$ (the cold inputs $[R'_{A,i}R'_{B,j}]\widetilde{G}^{(2)\prime}$ are unpredictable to $\mathcal{Z}$ before $\mathsf{Sim_{TP}}$ samples its randomness).

The post-corruption view in the ideal world is therefore identical to the real-world view, except with negligible probability.

Authenticity follows from the UC realization together with Lemma 13 via the standard reduction (Corollary 5): any forgery in the real world would yield a distinguisher between real and ideal executions. □

Remark 36. The theorem above is stated for a single arity-2 tensor gate. When several arity-2 gates share the same $\widetilde{G}^{(2)}$ (Section 4.3.1), indistinguishability across all such gates is established by the circuit-wide hybrid over the hot-carrying objects of the circuit (Lemma 14), under the joint assumption of Definition 42; the simulator samples $k^{(2)\prime}$ once and reuses it, as in Algorithm 14. We stress that the plain UC composition theorem is not directly applicable here, since the gates share the secret state $(\Delta, k^{(I)})$; see [CR03] for composition with joint state, and Theorem 6 below for the circuit-wide statement.

Circuit-wide authenticity. We now lift the single-object statements to the whole garbled circuit. Fix a circuit C and an honest-prover execution of the scheme on C. Call *help-carrying object* every instance, in C, of an arity-I tensor product ($2 \le I \le I_{\max}$), of π_{bin2hot}, or of π_{word2hot}; free gadgets (not, $\mathsf{hot2word}$, $\mathsf{hot2bin}$, and the free evaluation of Section 4.2) contribute no help and no X-entries. List the help-carrying objects $O_1, \ldots, O_{T_{\mathsf{hot}}}$ in a topological order of C. For each O_κ let $U_\kappa^\star$ be its unblinded hot input (Definition 44), and let P_C be the total number of X-entries of all objects (hot and non-hot). Write $\mathsf{sec} := \left(k^{(2)}, \ldots, k^{(I_{\max})}, K, k_1^{\mathsf{b}}, \ldots, k_l^{\mathsf{b}}\right)$ for the transmitted-material secrets of Algorithm 6, so that the transmitted group elements are exactly the set Mat of Definition 42. As in Lemma 12, q_X denotes the number of $\mathcal{G}_{\mathsf{RO}}^{X}$ queries issued during the session (without loss of generality all routed through the corrupt verifier $V^\star$, since $\mathcal{Z}$ and $\mathcal{A}$ communicate freely), and $q_X^{\mathcal{Z}}$ the number issued by $\mathcal{Z}$ before the prover samples its randomness.

Definition 45 (Protocol π_{circ})**.** On input x, the prover runs Gb (Algorithm 6) and En (Algorithm 3), posts the output-wire hashes to $\mathcal{G}_{\mathsf{trust}}$ as in Algorithm 6, and sends the garbled material, the garbled input $W(x)$ and the cleartexts to the verifier, who runs Ev (Algorithm 5) and outputs the garbled circuit output Y.

Definition 46 (Ideal functionality $\mathcal{F}_{\mathsf{circ}}$)**.** $\mathcal{F}_{\mathsf{circ}}$ proceeds as follows.

1. On $(\mathsf{Input}, \mathsf{sid}, x)$ from the prover: sample $\Delta \xleftarrow{\$} \mathbb{Z}_q^*$, all wire nonces, and sec internally; compute the garbled input $W(x)$, all transmitted generators, all helps, and the garbled values of every wire of C by the scheme equations (Theorems 1, 2, 3); record the garbled output Y.
2. Post the output-wire hashes $\mathsf{Hash}(w_o(\sigma))$, $o \in O$, $\sigma \in \{0,1\}$, to $\mathcal{G}_{\mathsf{trust}}$.
3. Send the transcript (material, $W(x)$, cleartexts, helps) and the per-object garbled outputs to the verifier, and leak them to the simulator.
4. On $(\mathsf{Verify}, \mathsf{sid}, Y')$: return 1 iff $Y' = Y$ for the recorded session. This interface is an ideal-world analysis device in the sense of $\mathcal{F}^{-\mathsf{Verify}}$; its real-world counterpart is the hash comparison against the commitments posted to $\mathcal{G}_{\mathsf{trust}}$, which is available to $\mathcal{Z}$ because $\mathcal{G}_{\mathsf{trust}}$ is global.

Definition 47 (Simulator $\mathsf{Sim}_{\mathsf{circ}}$)**.** $\mathsf{Sim}_{\mathsf{circ}}$ generalizes Algorithm 14 from one gate to C:

1. Sample fresh $\Delta' \xleftarrow{\$} \mathbb{Z}_q^*$, fresh input nonces, and fresh secrets sec' (each $k^{(I)\prime}$ and K' once per circuit, each $k_\iota^{\mathsf{b}\prime}$ once per gadget); compute all transmitted generators and the garbled input $W'(x)$ from the primed values.
2. Walk C in topological order. For each object O_κ: program $\mathcal{G}_{\mathsf{RO}}^X$, at every non-hot point of O_κ computed from the primed labels and generators, with a fresh $z' \xleftarrow{\$} \mathbb{Z}_q^*$; at the hot point, with $z'_\star := \Delta'^{-1} \cdot Y_\kappa^\star$, where $Y_\kappa^\star$ is the hot output value of O_κ delivered by $\mathcal{F}_{\mathsf{circ}}$; set the help $H'_\kappa := \Delta' \cdot \prod z'$ over all points of O_κ. The evaluator's reconstruction then returns the non-hot z' values and the delivered $Y_\kappa^\star$, and these determine the primed labels feeding the next objects, which $\mathsf{Sim}_{\mathsf{circ}}$ tracks.
3. Post to $\mathcal{G}_{\mathsf{trust}}$, for each output wire, the hash of the delivered output label together with the hash of a fresh uniform value in place of the never-revealed opposite label (Assumption 1).
4. Send the assembled transcript to $\mathcal{A}$.

Assumption 1 (Commitment hiding on uniform labels)**.** *For every PPT distinguisher, in an honest execution with $\Delta \xleftarrow{\$} \mathbb{Z}_q^*$ and uniform $\{R_i\}$ sampled by* Gb*, the hashes of the never-revealed opposite labels are indistinguishable from hashes of fresh uniform values given the rest of the view: $\{\mathsf{Hash}(w_i(z_i) \cdot \Delta^{1-2z_i})\}_i \approx_c \{\mathsf{Hash}(S_i)\}_i$ with $S_i \xleftarrow{\$} \mathbb{Z}_p^*$, with advantage $\varepsilon_{\mathsf{hide}}(\kappa) = \mathsf{negl}(\kappa)$. This holds unconditionally if* Hash *is modelled as a random oracle; for a standard-model* Hash *(Definition 40(3)) it is a mild additional hiding property, needed only because Algorithm 6 publishes commitments to* both *labels of the input and output wires, material that the per-object analysis of Lemmas 11 and 12 does not see.*

Lemma 14 (Circuit-wide Δ-hiding)**.** *Consider an honest-prover, corrupt-verifier execution of π_{circ} in the GUC-RO model. The view of $V^\star$ in which all helps and hot entries are computed with the sampled Δ is computationally indistinguishable from the view in which they are computed with an independent $\Delta' \xleftarrow{\$} \mathbb{Z}_q^*$, with distinguishing advantage at most*

$$\varepsilon_{\mathsf{hide}}(\kappa) \;+\; T_{\mathsf{hot}} \cdot q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) \;+\; \frac{2P_C\,(P_C + q_X^{\mathcal{Z}})}{q-1},$$

where $\mathsf{Adv}^{\Delta\mathsf{P}}$ refers to the joint assumption of Definition 42.

Proof. We refine the hybrid chain of Lemma 12, running it over all of C at once.

G_0 **(real)** is the real execution; G_1 answers all X-queries by lazy sampling, identically.

$G_{1.5}$ **(hide opposite labels).** Replace, in the material posted to $\mathcal{G}_{\mathsf{trust}}$, the hashes of the never-revealed opposite labels by hashes of fresh uniform values. By Assumption 1, $|\Pr[G_1 = 1] - \Pr[G_{1.5} = 1]| \le \varepsilon_{\mathsf{hide}}(\kappa)$. From here on the view contains no Δ-dependent information outside the transmitted generators, the hot wire labels, the helps and the hot entries.

G_2 **(re-parametrize all objects).** For every *distinct* group element arising as a non-hot point of some object, replace the oracle answer by a fresh independent value (literal duplicates, identical gates on identical wires, share one point and hence one value, coherently); for every object, set the hot output to $H_\kappa \cdot h_\kappa^{-1}$, which by Lemma 7 equals $\Delta \cdot X(U_\kappa^\star)$ and preserves the joint distribution. As in Lemma 12, this reassignment is sound unless (a) $\mathcal{Z}$ queried one of the P_C points before the prover sampled its randomness, probability at most $P_C\, q_X^{\mathcal{Z}}/(q-1)$ since every point involves at least one fresh uniform nonce; or (b) two of the prover's points coincide by chance: any two non-duplicate points are of the form $[\xi]B, [\xi']B'$ with $B, B' \in \mathsf{Mat} \cup \{G\}$ and ξ, ξ' distinct monomials in the independent uniform nonces (possibly times powers of Δ), so each pair collides with

probability at most $1/(q-1)$, giving at most $P_C^2/(q-1)$ in total; this includes the collisions of a hot point with a programmed non-hot point.

G_3 **(switch Δ).** Let $\mathsf{HotQuery}$ be the event that some session query to $\mathcal{G}_{\mathsf{RO}}^X$ hits some $U_\kappa^\star$. Conditioned on $\neg\mathsf{HotQuery}$, every hot entry equals $\Delta \cdot Z_\kappa$ with Z_κ uniform and unseen, hence is uniform and independent of Δ; all other view components are either fresh programmed values, values independent of Δ, or (after $G_{1.5}$) hashes of values independent of Δ; the helps are determined by the entries. Computing helps and hot entries with Δ' instead of Δ therefore leaves the view distribution unchanged: $|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \le \Pr[\mathsf{HotQuery}]$.

Bounding $\Pr[\mathsf{HotQuery}]$: one reduction to Definition 42. $\mathcal{B}$ receives (G, Mat) and plants it as the entire transmitted material of C; the labels are handled as in Step 1 of Lemma 11, circuit-wide: $\mathcal{B}$ samples all transmitted input labels directly (they are uniform; the nonces are fixed implicitly), and walks C in topological order maintaining the invariant that *every point the verifier can compute is a known scalar multiple of an element of* $\mathsf{Mat} \cup \{G\}$*, and every hot point $U_\kappa^\star$ is a known scalar multiple $[\sigma_\kappa]T_\kappa$ of the withheld target T_κ of O_κ's family*: for a tensor object the scalar is the product of the held hot labels (which carry the Δ-powers that convert $\widetilde{G}_0^{(I)}$ into $\widetilde{G}_{-I}^{(I)}$, as in Cases 1 and 3), for π_{bin2hot} likewise with $\widetilde{G}^{\mathsf{b}}_{-d_\iota}$, and for π_{word2hot} the scalar is $\Omega(a)$ with target $[K]G$ (Case 2). $\mathcal{B}$ programs every non-hot point with a fresh value, samples every hot output directly (uniform while its point is unqueried), sets the helps by the repair form, posts hashes of the labels it holds and of fresh uniforms for the opposite labels (matching $G_{1.5}$), and answers oracle queries by lazy sampling. The simulated view is that of G_2 up to the collision events already counted. $\mathcal{B}$ guesses $\iota \xleftarrow{\$} \{1, \ldots, q_X\}$ and $\kappa \xleftarrow{\$} \{1, \ldots, T_{\mathsf{hot}}\}$ at the outset and outputs $[\sigma_\kappa^{-1}]Q_\iota$, where Q_ι is the ι-th query falling outside the programmed points. If $\mathsf{HotQuery}$ occurs, the first offending query determines a correct pair, guessed with probability at least $1/(q_X T_{\mathsf{hot}})$, in which case $\mathcal{B}$ outputs the withheld element T_κ and wins the joint game. Hence $\Pr[\mathsf{HotQuery}] \le T_{\mathsf{hot}}\, q_X\, \mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) + P_C^2/(q-1)$, the slack reusing the collision count of G_2 (a hot point colliding with a programmed one, or two hot points colliding).

Summing the hops and absorbing the statistical terms into $2P_C(P_C + q_X^{\mathcal{Z}})/(q-1)$ proves the lemma. □

Theorem 6 (Circuit-wide UC authenticity)**.** *In the honest-prover, corrupt-verifier scenario with static corruptions, π_{circ} securely realizes $\mathcal{F}_{\mathsf{circ}}^{-\mathsf{Verify}}$ in the $(\mathcal{G}_{\mathsf{trust}}, \mathcal{G}_{\mathsf{RO}}^X)$-hybrid GUC-RO model, with simulator $\mathsf{Sim}_{\mathsf{circ}}$ and distinguishing advantage at most*

$$\varepsilon_{\mathsf{hide}}(\kappa) \;+\; T_{\mathsf{hot}} \cdot q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) \;+\; \frac{3P_C\,(P_C + q_X^{\mathcal{Z}})}{q-1}.$$

Proof. We compare three experiments.

H_0 **(real).** $V^\star$ receives the transcript computed under $(\Delta, \{R\}, \mathsf{sec})$ and evaluates.

H_1 **(decoupled hot outputs).** As H_0, except that every object's hot output is decoupled from the oracle: the hot entry is set to $\Delta \cdot Z_\kappa$ with fresh uniform Z_κ, and the helps are adjusted by the repair form. By Lemma 14 (whose hybrids $G_{1.5}$-G_3 effect exactly this decoupling, and whose Δ-switch is not yet needed), $|\Pr[H_0 = 1] - \Pr[H_1 = 1]|$ is at most the lemma's bound.

H_2 **(ideal).** $\mathcal{F}_{\mathsf{circ}}$ computes all garbled values under its internal $(\Delta, \{R\}, \mathsf{sec})$ and delivers the per-object outputs; $\mathsf{Sim}_{\mathsf{circ}}$ builds the transcript under independent $(\Delta', \{R'\}, \mathsf{sec}')$ and programs $\mathcal{G}_{\mathsf{RO}}^X$ so that the evaluator's reconstruction returns the delivered values (Definition 47). In H_1, the view consists of a consistently-parametrized transcript sample together with hot values that are uniform and independent of everything else, helps being determined; in H_2, the view has exactly the same form, the transcript is a fresh consistent sample under the primed parameters, and the delivered hot values $Y_\kappa^\star = \Delta \cdot X_{\mathcal{F}}(U_\kappa^\star)$ are

uniform and independent from $V^\star$'s side, the forced programming $z'_\star = \Delta'^{-1} Y^\star_\kappa$ making the reconstruction consistent. The two distributions coincide except when a $\mathsf{Program}$ call fails because $\mathcal{Z}$ pre-queried one of the P_C primed points, probability at most $P_C\, q_X^{\mathcal{Z}}/(q-1)$.

The triangle inequality gives the stated bound. Corruption is static: the honest prover is never corrupted mid-session (see the remark below), and a corrupt verifier holds no secrets, so no post-corruption consistency argument is required. □

Corollary 4 (Authenticity of the full scheme)**.** *In the honest-prover, corrupt-verifier scenario, π_{circ} satisfies authenticity in the sense of Definition 4: every PPT $V^\star$ outputs $Y' \neq Y$ accepted by the hash comparison against the $\mathcal{G}_{\mathsf{trust}}$-posted commitments with probability at most the bound of Theorem 6 plus $\varepsilon_{\mathsf{pre}}(\kappa) + \varepsilon_{\mathsf{cr}}(\kappa)$.*

Proof. The adjudication predicate required by Lemma 9 is available to $\mathcal{Z}$ in the real world: $\mathcal{G}_{\mathsf{trust}}$ is global, so $\mathcal{Z}$ can compare $\mathsf{Hash}(Y'_o)$ against the posted values. In the ideal world $\mathcal{F}_{\mathsf{circ}}$ rejects every $Y' \neq Y$, and an accepted real-world forgery must either match a posted hash with a different preimage ($\varepsilon_{\mathsf{pre}} + \varepsilon_{\mathsf{cr}}$) or exhibit the recorded opposite label, in which case the environment distinguishing real from ideal contradicts Theorem 6. □

Remark 37 (Composition with cut-and-choose). With per-copy independent correlations and secrets $(\Delta_j, \mathsf{sec}_j)$ and per-copy committed labels in π_{GC}, the verifier's view across the s copies is a product of independent instances: the opened copies are simulated trivially from their revealed secrets, and each unopened copy contributes one instance of Theorem 6, for a total additive loss of at most $(s-t)$ times the theorem's bound. No shared state survives across copies, so no joint-state argument is needed at that level.

Remark 38 (Why static corruption). Adaptive corruption of the prover cannot be simulated for π_{circ} as constructed: Algorithm 6 posts *binding* commitments $\mathsf{Hash}(w_o(\sigma))$ to both labels of every output wire before any corruption occurs, Hash is not programmable (Definition 40(3)), and the model excludes secure erasures (Definition 39). A simulator that reveals primed randomness $(\Delta', \{R'\}, \mathsf{sec}')$ upon corruption cannot make the pre-posted hash of the opposite label $w_o(1-z) = Y_o \cdot \Delta^{\pm 1}$ consistent with Δ'. This obstruction is absent for the stand-alone gate protocols π_f and π_{TP}, which post no commitments, so the adaptive claims of Theorems 4 and 5 are unaffected as stated; at circuit level, static corruption is the honest limit of the construction.

Remark 39. The authenticity property for the evaluation of $f : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_d$ follows as the special case of Theorem 6 for the two-object circuit consisting of the tensor product followed by the free projection $\check{f}$; no separate composition argument is needed.

Corollary 5. *In the honest-prover, malicious-verifier setting, protocol π_{TP} satisfies authenticity in the sense of Definition 4 for the tensor product output $T_{A\otimes B}(a,b)$.*

Proof. By Lemma 13, any ideal execution with $\mathcal{F}_{\mathsf{TP}}$ has perfect authenticity. By Theorem 5 and the UC realization criterion (Definition 39), the real execution of π_{TP} is indistinguishable from the ideal one. A non-negligible authenticity break in the real world would thus yield a distinguisher between the two, contradicting UC security.

In the real execution, $\mathcal{Z}$ adjudicates forgery success by comparing Hash of the candidate output labels against the output-wire commitments that Gb posts to $\mathcal{G}_{\mathsf{trust}}$ when the gate is deployed in a garbled circuit, the one-object instance of Theorem 6. The stand-alone protocol posts no commitments (Remark 38), which is precisely what permits its adaptive-corruption claim; since no corruption occurs inside the authenticity experiment itself, the committed adjudication and the adaptive claim never apply to the same execution. The comparison is readable by $\mathcal{Z}$ because $\mathcal{G}_{\mathsf{trust}}$ is global, and realizes the acceptance check of Definition 4 without access to the prover's secrets. □

5.2.2 The not and bin2hot functionalities

The not functionality implements, at the garbled level, the "opposite" operation introduced in Remark 24: for a Boolean wire $w(z) = R\Delta^z$, the cleartext negation $\mathsf{NOT}(z) = 1 - z$ corresponds to mapping the label $w(z)$ to its opposite label $w(z)^{-1}$. Any forgery would require producing the unrevealed opposite label, which is unpredictable given the verifier's view. For a Boolean wire w_i with garblings $w_i(0) = R_i$ and $w_i(1) = R_i\Delta$, the NOT gate computes $f(z) = 1 - z$, producing the output garbling $w_o(1 - z) = w_i(z) \cdot \Delta^{1-2z}$ without additional ciphertext. A corrupt verifier holds only $w_i(z) = R_i\Delta^z$; producing $w_o(1 - z')$ for $z' \neq z$ requires the value $R_i\Delta^{1-z}$, which is a uniform element of $\mathbb{Z}_q^*$ independent of her view, and is therefore guessed with probability at most $1/(q - 1)$. The output-wire commitment adds a preimage-resistance term $\varepsilon_{\mathsf{pre}}(\kappa)$. Thus, the NOT gate preserves authenticity in the game-based sense of Definition 4, with advantage $\mathsf{Adv}(\kappa) \leq 1/(q - 1) + \varepsilon_{\mathsf{pre}}(\kappa)$. No elliptic-curve operation occurs in this gadget, so no group-hardness assumption is involved. From here, and by combining it with the authenticity of the tensor product $T_{A\otimes B}$, the bin2hot functionality of Section 4.4.1 also has the authenticity property in the same game-based sense. A fully formal UC treatment would introduce dedicated ideal functionalities for not and bin2hot and then appeal to the UC composition theorem [Can20]; we omit these details for brevity.

Algorithm 15 Protocol π_{bin2hot}: Garbling and Evaluation of $\mathsf{bin2hot} : \{0,1\}^d \to \mathbb{Z}_n$

Require: Garbled bits $\{w_{i_j}(a_j) = R_{i_j}\Delta^{a_j}\}_{j\in\mathbb{Z}_d}$ with $d = \lceil \log_2 n \rceil$ and $a = \sum_j a_j 2^j \in \mathbb{Z}_n$.
Per-gadget secret $k^{\mathsf{b}} \xleftarrow{\$} \mathbb{Z}_q^*$ with $\widetilde{G}^{\mathsf{b}} := [k^{\mathsf{b}}]G$; transmitted generators $\widetilde{G}^{\mathsf{b}}_{-c} := [k^{\mathsf{b}}\Delta^{-c}]G$ for $c \in \{0, \ldots, d-1\}$ (the power $c = d$ is withheld). Standing: $\kappa, p, E, G, \Delta, X$.

Ensure: Garbled output $W_I(a) = (R_{I,k}\Delta^{\delta_k(a)})_{k\in\mathbb{Z}_n}$.

Prover (Garbler).

1: Compute help H per equation (6) (with base $\widetilde{G}^{\mathsf{b}}$); send $\{w_{i_j}(a_j)\}_j$, cleartexts $\{a_j\}_j$, generators $\{\widetilde{G}^{\mathsf{b}}_{-c}\}_{c\in\{0,\ldots,d-1\}}$, and H to the verifier.

Verifier (Evaluator).

2: For each $k \in \mathbb{Z}_n$ with $k \neq a$, compute w_k as the mixed-hot tensor entry per Definition 27. Set $w_a := 1$ and reconstruct the hot entry as $W_I(a)_a := H \cdot \left(\prod_{k\neq a} w_k\right)^{-1}$ in $\mathbb{Z}_q^*$.

3: **return** $W_I(a) = (W_I(a)_k)_{k\in\mathbb{Z}_n}$.

5.2.3 The word2hot functionality

This is the protocol π_{word2hot} for the word2hot functionality (Section 4.4.4), which converts a word wire garbling $\Omega(a) = r\Delta^a \in \mathbb{Z}_p^*$ into a one-hot vector garbling $W_I(a)_j = R_{i_j}\Delta^{\delta_j(a)}$ for $a \in \mathbb{Z}_n$. The protocol preserves game-based authenticity for the one-hot output: intuitively, a corrupt verifier cannot forge a valid one-hot output for $a' \neq a$ without either recovering Δ (breaking ECDLP) or guessing fresh nonces. The pseudorandomness of X (Proposition 2) secures the intermediate wires. We do not introduce a separate UC functionality for word2hot; we treat its security in this stand-alone, game-based sense.

Algorithm 16 Protocol π_{word2hot}: Garbling and Evaluation of $\mathsf{word2hot} : \mathbb{Z}_n \to \mathbb{Z}_n$

Require: Word garbling $\Omega(a) = r\Delta^a$ for cleartext $a \in \mathbb{Z}_n$. Standing: $\kappa, p, E, G, \Delta, X$.
Ensure: Garbled output $W_I(a) = (R_{i_j}\Delta^{\delta_j(a)})_{j\in\mathbb{Z}_n}$.

Prover (Garbler).

1: Use the circuit-wide secret $K \xleftarrow{\$} \mathbb{Z}_q^*$ sampled by Gb (Algorithm 6); set $G' := [K]G$ and $G'_i := [\Delta^i]G'$ for $i \in \{\pm 1, \ldots, \pm(n-1)\}$. Set output nonces $R_{i_j} := X([r\Delta^j]G')$ and compute help $H := \Delta \prod_{j=0}^{n-1} X([r\Delta^j]G')$.
2: Send $(\Omega(a), a, \{G'_i\}_i, H)$ to verifier.

Verifier (Evaluator).

3: For each $j \in \mathbb{Z}_n$, such that $j \neq a$, compute $W_I(a)_j := X([\Omega(a)]G'_{j-a})$. For $j = a$, set $W_I(a)_a := H \cdot \left(\prod_{j\neq a} W_I(a)_j\right)^{-1}$ in $\mathbb{Z}_q^*$.
4: Output $W_I(a) = (W_I(a)_j)_{j\in\mathbb{Z}_n}$.

The protocol preserves game-based authenticity for the one-hot output: intuitively, forging a valid output for $a' \neq a$ would again require computing Δ or predicting new nonces. The pseudorandomness of X (Proposition 2) secures intermediate wires. We do not introduce a separate UC functionality for $\mathsf{word2hot}$.

5.2.4 Multiplication of large numbers

This protocol computes $M(a, b) = ab \pmod m$ for large integers $a, b \in \mathbb{Z}_m$ with $m = 2^{31}-1$, per Remark 25 in the previous section. The gate uses discrete logarithms to achieve $O(m)$ complexity, requiring only one ciphertext.

Algorithm 17 Protocol π_{mult}: Product Garbling and Evaluation Gate for Large Integers

Require: Multiplication gate $M(a, b) = ab \pmod m$ over Mersenne prime $m = 2^{31} - 1$, with generator $g = 2$ of $\mathbb{Z}_m^*$ and discrete-log table $l : \mathbb{Z}_m^* \to \mathbb{Z}_m$ ($2^{l(x)} = x \pmod m$), $l(0) := 2m - 3$). Garbled inputs $W_A(a), W_B(b)$ over wire sets of size m. Standing parameters: $\kappa, p, E, G, \Delta, X, G', \{G'_i\}$ (Section 4).
Ensure: Garbled output $W_O(ab) = (R_{O,i}\Delta^{\delta_i(ab)})_{i\in\mathbb{Z}_m}$.

Prover (Garbler).

1: Compute help $H = \Delta \prod_{j=0}^{4m-6} X([r\Delta^j]G')$ for the embedded $\mathsf{word2hot}$ step over $\mathbb{Z}_{4m-5}$.

Verifier (Evaluator). Pipeline (each step uses the named functionality from the indicated section):

2: Execute steps (2)-(6) of Remark 25's pipeline: basic functionality with l; $\mathsf{hot2word}$ and multiplication to $\Omega(\sigma) = r_A r_B \Delta^\sigma$, $\sigma := l(a) + l(b)$; $\mathsf{word2hot}$ over $\mathbb{Z}_{4m-5}$ (Algorithm 16); and the basic functionality with e, obtaining $W_O(ab)$.

The protocol preserves authenticity for the product gate in a game-based sense: intuitively, a corrupt verifier cannot forge a different valid garbled product $W_O(ab')$ for $ab' \neq ab \pmod m$ without either guessing fresh nonces or breaking the Δ-power assumption (Definition 41) underlying the embedded $\mathsf{word2hot}$ step. By modularity, this game-based authenticity extends to circuits that use the multiplication gate, but we do not formalize a dedicated UC functionality for it.

5.2.5 Verification protocol π_{ver}

The protocol π_{ver} for the verification function $V : \mathbb{Z}_m \to \mathbb{Z}_2$ follows, which verifies whether the garbled output $W_C(c')$ of circuit C matches the claimed value $c \in \mathbb{Z}_m$.

Definition 48. A *ledger functionality* $\mathcal{L}$ is an ideal functionality parameterized by a liveness parameter $\Delta_{\mathsf{live}} \in \mathbb{N}$. Parties may submit transactions to $\mathcal{L}$ at any point in the execution. A submitted transaction TX becomes *confirmed* no later than Δ_{live} time-units after submission, except with probability $\mathsf{negl}(\kappa)$ in the security parameter. Confirmed transactions are immutable and publicly readable. Each transaction can carry conditional unlocking clauses (preimage requirements, timelocks $t \geq t_0$, or conjunctions thereof) and may consume previously-confirmed outputs according to those clauses.

Definition 49. Consider the stand-alone execution of the verification protocol π_{ver} (Algorithm 18) between a prover P and an evaluator V, interacting with a ledger functionality $\mathcal{L}$ (Definition 48). Fix a function $F : \mathbb{Z}_n \to \mathbb{Z}_m$ and an input $x \in \mathbb{Z}_n$, and let $z^\star = F(x)$ be the correct output. Let $\mathsf{ver} : \mathbb{Z}_m \to \{0,1\}$ be the verification predicate implemented by π_{ver}, where $\mathsf{ver}(z^\star) = 1$ if and only if the intended success condition for $F(x)$ holds.

The protocol uses two transactions submitted to $\mathcal{L}$: (1) TX_1, the *challenge transaction*, submitted by V, with an unlocking clause requiring a preimage r such that $\mathsf{Hash}(r) = h_f$, where h_f is a public commitment established by π_{ver}, and (2) TX_2, the *timeout transaction*, submitted by P, with an unlocking clause requiring time $t \geq t_{\mathsf{lock}}$ for some configured timelock parameter.

We model the ledger's final state by a bit $\mathsf{pay} \in \{0,1\}$ which is 1 if $\mathcal{L}$ confirms TX_2, and 0 if $\mathcal{L}$ confirms TX_1 (mutually exclusive by construction; the underlying conditional outputs are single-spend).

We say that π_{ver} is *fair against a malicious prover* if for every PPT adversary $\mathcal{A}$ controlling P it holds that $\Pr\left[\,\mathsf{pay} = 1 \ \wedge\ \mathsf{ver}(z^\star) = 0\,\right] \leq \mathsf{negl}(\kappa)$, where the probability is over the randomness of $\mathcal{A}$, the honest party V, and the randomness of $\mathcal{L}$. In words: except with negligible probability, the ledger never confirms a successful claim for P when the underlying cleartext computation $F(x)$ does not satisfy the verification predicate.

Algorithm 18 Protocol π_{ver}: Garbling and Verification of $\mathsf{ver} : \mathbb{Z}_m \to \mathbb{Z}_2$

Require: Standing parameters: κ, p, Δ. Circuit C with output wires $W_C = (W_C(c')_i)_{i \in \mathbb{Z}_m}$ for cleartext $c' \in \mathbb{Z}_m$, claimed value $c \in \mathbb{Z}_m$, output-wire nonces $R_{C,i} \in \mathbb{Z}_p^*$ for W_C, verification output wire w_{ver}, hash function $H : \mathbb{Z}_p \to \{0,1\}^\lambda$. Ledger functionality $\mathcal{L}$ (Definition 48) with timelock parameter t_{lock}.

Ensure: Garbled output $w_{\mathsf{ver}}(z) = R_f \Delta^z$, where $z = V(c') = \delta_c(c') \in \{0,1\}$ (that is, $z = 1$ iff $c' = c$), and $R_f = R_{C,c}$. Ledger outcome $\mathsf{pay} \in \{0,1\}$.

Prover (Garbler).

1: Set output nonce $R_f := R_{C,c}$ and compute commitment $h_f := \mathsf{Hash}(R_f)$.
2: Submit transaction TX_2 to $\mathcal{L}$ unlocking under condition $t \geq t_{\mathsf{lock}}$ (timeout claim).
3: Send $W_C(c') = (R_{C,i}\Delta^{\delta_i(c')})_{i \in \mathbb{Z}_m}$ and cleartext c' to verifier.

Verifier (Evaluator). The verifier reads the single coordinate $W_C(c')_c$ of the one-hot output vector W_C over $\mathbb{Z}_m$; this coordinate is hot ($= R_{C,c}\Delta$) iff $c' = c$ and cold ($= R_{C,c} = R_f$) iff $c' \neq c$, which is exactly the Boolean equality predicate $z = \delta_c(c')$.

4: Compute $z := V(c') = \delta_c(c')$ and $w_{\mathsf{ver}}(z) := W_C(c')_c$.
5: **if** $z = 0$ (that is, $c' \neq c$) **then** set $r := w_{\mathsf{ver}}(0) = R_f$ and submit transaction TX_1 to $\mathcal{L}$ unlocking under condition $\mathsf{Hash}(r) = h_f$ (challenge claim).
6: **end if**

Ledger.

7: $\mathcal{L}$ confirms exactly one of TX_1 (if submitted before t_{lock}) or TX_2 (if TX_1 is not confirmed by t_{lock}). Set $\mathsf{pay} := 1$ if TX_2 is confirmed, else 0.

The protocol achieves a stand-alone (game-based) authenticity guarantee: given the hash commitment $h_f = \mathsf{Hash}(R_f)$ embedded in the unlocking clause of TX_1, a cheating party can cause an inconsistent ledger claim only with negligible probability, under the

ECDLP assumption (Assumption 34) and the preimage resistance of H. Preimage resistance of H is the standard-model property of Definition 40(3); no random-oracle modelling of H is used. We do not introduce a dedicated UC functionality for ledger-based verification; our guarantees for π_{ver} are stand-alone and game-based, parameterized by the ledger $\mathcal{L}$.

We assume that the commitment h_f submitted to $\mathcal{L}$ is bound to the committed label hash for the verifier wire $w_v(0)$ posted to $\mathcal{G}_{\mathsf{trust}}$, with $\mathsf{Hash}(w_v(0))$ validated via $\mathcal{G}_{\mathsf{trust}}$.

Assumption 2. *The conditional output consumed by π_{ver} is constructed so that the timeout clause of TX_2 becomes satisfiable only if, before a delivery deadline $t_{\mathsf{deliver}} < t_{\mathsf{lock}} - \Delta_{\mathsf{live}}$, the prover has published (or delivered, with publicly checkable receipts via $\mathcal{G}_{\mathsf{trust}}$) garbled inputs and circuit material passing all validations of Algorithms 1 and 2; otherwise a default clause spendable by V activates at t_{deliver}. In particular, if the honest evaluator outputs $\perp$ because of a failed validation or missing delivery, then $\mathsf{pay} = 0$.*

Theorem 7. *Assume that the hash function H used in π_{ver} is preimage-resistant and collision-resistant, that the garbled evaluation of F is sound against a malicious prover in the sense of Definition 5 (Corollary 6), and that Assumption 2 holds. Then, against a malicious prover, π_{ver} satisfies fairness in the sense of Definition 49.*

Proof. Fix any PPT adversary $\mathcal{A}$ controlling P and an input x with correct output $z^\star = F(x)$. By Definition 49, unfairness corresponds to the event that the ledger executes the "success" branch ($\mathsf{pay} = 1$) while $\mathsf{ver}(z^\star) = 0$.

In protocol π_{ver}, the ledger moves to the success branch ($\mathsf{pay} = 1$) if, and only if, the prover successfully confirms the timeout transaction TX_2 after the timelock, and no valid challenge transaction TX_1 consuming the same conditional output was confirmed before the timelock. Let t_{lock} denote the absolute time at which TX_2 becomes confirmable (that is, the timelock encoded in the conditional output consumed by π_{ver}), and let $t_{\mathsf{broadcast}}$ denote the time at which the verifier V broadcasts the challenge transaction TX_1. We assume π_{ver} is parameterized so that $t_{\mathsf{lock}} > t_{\mathsf{broadcast}} + \Delta_{\mathsf{live}}$, where Δ_{live} is the ledger's liveness parameter (Definition 48). Under this assumption, if V holds the preimage R_f at time $t_{\mathsf{broadcast}}$, then by the liveness property of $\mathcal{L}$, TX_1 is confirmed by time t_{lock} except with probability $\mathsf{negl}(\kappa)$.

We distinguish two cases. *Abort case.* If the honest evaluator outputs $\perp$ (failed validation or missing delivery), then by Assumption 2 the default clause spendable by V activates at t_{deliver} and $\mathsf{pay} = 0$; no unfairness arises.

Non-abort case. Otherwise the evaluator completed all validations of Algorithms 1 and 2. By soundness against a malicious prover (Definition 5, Corollary 6), the decoded cleartext output equals $F(x) = z^\star$ except with negligible probability. Suppose $\mathsf{ver}(z^\star) = 0$. Then the coordinate read in line 4 is cold, so V obtains $r = w_{\mathsf{ver}}(0) = R_f$. Since the commitment h_f is bound via $\mathcal{G}_{\mathsf{trust}}$ to the committed label hash $\mathsf{Hash}(w_v(0))$, a prover submitting $h_f \neq \mathsf{Hash}(R_f)$ is caught during validation, reducing to the abort case; and exhibiting $r' \neq R_f$ with $H(r') = h_f$ would contradict collision resistance. V broadcasts TX_1 at $t_{\mathsf{broadcast}} < t_{\mathsf{lock}} - \Delta_{\mathsf{live}}$; by the liveness of $\mathcal{L}$, TX_1 is confirmed before t_{lock} except with probability $\mathsf{negl}(\kappa)$, hence $\mathsf{pay} = 0$.

Combining the two cases, $\Pr[\mathsf{pay} = 1 \wedge \mathsf{ver}(z^\star) = 0] \leq \mathsf{negl}(\kappa)$, bounded by the advantages against soundness and the collision resistance of H, plus the ledger's liveness failure. Hence π_{ver} is fair in the sense of Definition 49. □

5.3 UC correctness in the honest verifier, corrupt prover scenario

This subsection proves that the OHMG protocol π_{GC} securely realizes the ideal functionality $\mathcal{F}_{\mathsf{GC}}$ with UC correctness in the honest verifier, corrupt prover scenario with static corruption of the prover (Remark 40), leveraging the cut-and-choose technique and $\mathcal{G}_{\mathsf{trust}}$'s ideal functionality, as defined by consistency and correctness in 3.

Remark 40. In the analysis of protocol π_{GC} we consider an honest evaluator V and a potentially malicious prover $P^\star$. We work with *static corruption* of the prover: $P^\star$ is corrupt from the outset of the execution. This matches the standard cut-and-choose setting: the binding property of the commitments, together with the consistency checks on the opened subset, prevents a corrupt prover from changing its garblings retroactively. Adaptive corruption of an initially honest prover would require the simulator to emulate honest garbling and hand over a consistent state on corruption; we do not claim it here.

Remark 41. For protocols π_f and π_{TP} (Theorems 4 and 5), the corruption model is unrestricted: the environment $\mathcal{Z}$ may corrupt the prover at any point in the execution. Pre-message corruption is handled by the simulator sampling its randomness on the spot (consistent with an empty transcript); post-message corruption is handled as in line 4 / line 9 of the respective simulators. There is no commitment phase to constrain the corruption window.

Algorithm 19 Protocol π_{GC}: Garbling and Evaluation of Circuit C Computing F

Require: Circuit C for $F : \mathbb{Z}_n \to \mathbb{Z}_m$, cleartext input $x \in \mathbb{Z}_n$ in binary form, statistical security parameter s (e.g. $s = 160$ for soundness error 2^{-41}, cf. Theorem 8). Standing parameters: κ, p, E, G (Section 4); each copy $j \in [s]$ carries its own independent correlation Δ_j.

Ensure: Garbled input $B_I(x)$, garbled output $W_O(F(x))$, cleartext output y.

Prover (Garbler).

1: Run $\mathsf{Gb}(1^\kappa, C)$ (Algorithm 6) to obtain s garblings $GC_1, \ldots, GC_s$, with input/output wire labels committed via $\mathcal{G}_{\mathsf{trust}}$ (Section 2.5.1). Send $\{GC_j\}_{j=1}^s$ to V.

2: For each $j \in [s]$, run $\mathsf{En}(f_e^{(j)}, x)$ to encode x as $B_I^{(j)}(x) = (R_{A,i,j}\Delta_j^{x[i]})_{i\in\mathbb{Z}_d}$; send $\{B_I^{(j)}(x)\}_{j\in[s]}$ and cleartext x to V.

Cut-and-choose ZKP.

3: V samples $J \subset [s]$, $|J| = s/2$, uniformly and sends to P; P opens each GC_j, $j \in J$, by revealing the garbling randomness ρ_j. V recomputes $\mathsf{Gb}(1^\kappa, C; \rho_j)$, aborts unless it equals the received GC_j *and* its input/output labels match the hashes committed via $\mathcal{G}_{\mathsf{trust}}$.

Verifier (Evaluator).

4: For each $j \notin J$, evaluate GC_j on $B_I^{(j)}(x)$ to obtain $W_O^j(F(x))$ and decode via De (using $\mathcal{G}_{\mathsf{trust}}$ for hash matching) to cleartext y^j.

5: Output the majority value y across $\{y^j\}_{j\notin J}$, or abort if no majority.

Algorithm 20 Ideal $\mathcal{F}_{\mathsf{GC}}$: Garbling and Evaluation of Circuit C Computing F

Require: Security parameter κ, prime $p \approx 2^{\kappa}$, circuit C computing $F : \mathbb{Z}_n \to \mathbb{Z}_m$, input wire set $A = \{a_0, \ldots, a_{d-1}\}$, output wire set $O = \{o_0, \ldots, o_{m-1}\}$.
Ensure: Garbled input $B_I(x)$, garbled output $W_O(F(x))$, cleartext output y.
Initialization
1: Choose global correlation $\Delta \in \mathbb{Z}_q^*$ (secret) and nonces $R_{A,i}, R_{O,j} \in \mathbb{Z}_p^*$ for each $a_i \in A$, $o_j \in O$, known only to trusted party.
On input $x \in \mathbb{Z}_n$ from prover P (originating from the environment $\mathcal{Z}$) in binary form $x[0], \ldots, x[d-1]$:
2: Compute garbled input $B_I(x) = (b_{A,i}(x[i]))_{i \in \mathbb{Z}_d}$, where $b_{A,i}(x[i]) = R_{A,i}\Delta^{x[i]}$ for each bit $x[i] \in \{0, 1\}$.
3: Send $B_I(x)$ and cleartext x to verifier V.
On request from verifier V to evaluate F:
4: Compute garbled output $W_O(F(x)) = (w_{O,j}(F(x)_j))_{j \in \mathbb{Z}_m}$, where $w_{O,j}(F(x)_j) = R_{O,j}\Delta^{\delta_j(F(x))}$.
5: Decode $W_O(F(x))$ to cleartext y using stored nonces and Δ.
6: Send $W_O(F(x))$ and y to V.

We stress that $\mathcal{F}_{\mathsf{GC}}$ sends both the clear input x and the clear output $y = F(x)$ to the verifier V, in addition to the corresponding garbled output vector $W_O(F(x))$. This reflects our explicit design choice not to target input or output privacy (Remark 3): $\mathcal{F}_{\mathsf{GC}}$ only enforces correctness and authenticity of the reported output.

Lemma 15. *Let $\mathcal{F}_{\mathsf{GC}}$ be the ideal functionality of Algorithm 20. In any ideal execution with honest evaluator V and arbitrary (possibly malicious) prover $P^\star$, the output y_V of V is either $\perp$ or the correct value $F(x)$. In particular, a malicious prover cannot cause V to accept an incorrect value $y_V \notin \{\perp, F(x)\}$.*

Proof. By inspection of Algorithm 20, $\mathcal{F}_{\mathsf{GC}}$ maintains an internal correlation Δ and nonce sets $\{R_{A,i}\}$, $\{R_{O,j}\}$, and on input x from the environment it always computes the output vector $W_O(F(x))$ according to the specification of F. The evaluator V only receives $W_O(F(x))$ (and possibly x), and no other value is ever delivered that could decode to a different output y_V. Hence, regardless of $P^\star$'s behaviour, V's output is either this prescribed value $F(x)$ or $\perp$ if the protocol aborts. □

Algorithm 21 Ideal-World Simulator $\mathsf{Sim}_{\mathsf{GC}}$, honest verifier

Require: Hybrids $\mathcal{G}_{\mathsf{trust}}, \mathcal{F}_{\mathsf{GC}}$; adversary $\mathcal{A}$ (corrupt prover).
Ensure: View indistinguishable from real-world π_{GC}.
1: **Emulate honest V:** accept $\mathcal{A}$'s prover messages (commitments, s garbled circuits, x, $B_I(x)$); sample $J \subset [s]$ uniformly with $|J| = s/2$, send to $\mathcal{A}$. On openings, validate via the recompute-and-compare check of line 3 and $\mathcal{G}_{\mathsf{trust}}$ (Algorithm 2); abort on failure.
2: **Output:** forward x to $\mathcal{F}_{\mathsf{GC}}$, receive $(W_O(F(x)), y)$; $\mathcal{F}_{\mathsf{GC}}$ delivers y to the honest verifier's dummy party.

Theorem 8. *Let π_{GC} be the protocol for garbling and evaluating the circuit C computing $F : \mathbb{Z}_n \to \mathbb{Z}_m$ as defined in Algorithm 19, and let $\mathcal{F}_{\mathsf{GC}}$ be the corresponding ideal functionality as defined in Algorithm 20. Consider the $(\mathcal{G}_{\mathsf{trust}}, \mathcal{G}_{\mathsf{RO}}^X)$-hybrid model with an honest evaluator V and a potentially malicious prover $P^\star$ (with static corruption of P, cf. Remark 40). For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ controlling $P^\star$ and any PPT environment $\mathcal{Z}$, the real-world execution $\mathsf{REAL}_{\mathcal{A},\pi_{\mathsf{GC}}}(\mathcal{Z})$ is computationally indistinguishable from the ideal-world execution $\mathsf{IDEAL}_{\mathsf{Sim}_{\mathsf{GC}},\mathcal{F}_{\mathsf{GC}}}(\mathcal{Z})$, where $\mathsf{Sim}_{\mathsf{GC}}$ is the simulator defined in*

Algorithm 21. The distinguishing advantage $\mathsf{Adv}_{\mathcal{Z}}(\kappa)$ *is negligible in the security parameter* κ*. In particular,* π_{GC} *securely realizes* $\mathcal{F}_{\mathsf{GC}}$ *with UC correctness in the honest-verifier, corrupt-prover scenario.*

Proof. Fix any PPT adversary $\mathcal{A}$ controlling the prover $P^\star$ and any PPT environment $\mathcal{Z}$. We prove that $\mathsf{REAL}_{\mathcal{A},\pi_{\mathsf{GC}}}(\mathcal{Z}) \approx_c \mathsf{IDEAL}_{\mathsf{Sim}_{\mathsf{GC}},\mathcal{F}_{\mathsf{GC}}}(\mathcal{Z})$ in the $(\mathcal{G}_{\mathsf{trust}}, \mathcal{G}_{\mathsf{RO}}^X)$-hybrid model, where $\mathsf{Sim}_{\mathsf{GC}}$ is Algorithm 21 (emulating the honest verifier V). We define hybrids H_0, H_1, H_2:

- H_0 **(real execution).** This is the real execution $\mathsf{REAL}_{\mathcal{A},\pi_{\mathsf{GC}}}(\mathcal{Z})$ with honest verifier V running π_{GC} and prover controlled by $\mathcal{A}$.
- H_1 **(output switch to** $\mathcal{F}_{\mathsf{GC}}$ **upon acceptance).** This hybrid is identical to H_0 up to and including: (i) the prover's messages, (ii) the verifier's sampling of the cut-and-choose challenge subset $J \subseteq [s]$ of size $|J| = s/2$, and (iii) all verifier-side validation checks (using $\mathcal{G}_{\mathsf{trust}}$ for commitment consistency for the cut-and-choose/ZK check). If the verifier aborts in H_0, then H_1 aborts as well and outputs $\perp$. Otherwise, H_1 does *not* compute the output by locally evaluating and decoding the unopened garbled circuits. Instead, it sends the clear input x (as received in the transcript) to $\mathcal{F}_{\mathsf{GC}}$ as the prover's input, requests evaluation, receives $(W_O(F(x)), y)$ from $\mathcal{F}_{\mathsf{GC}}$, and outputs y.
- H_2 **(ideal execution).** This is the ideal execution $\mathsf{IDEAL}_{\mathsf{Sim}_{\mathsf{GC}},\mathcal{F}_{\mathsf{GC}}}(\mathcal{Z})$, where $\mathsf{Sim}_{\mathsf{GC}}$ emulates the honest verifier: it interacts with $\mathcal{A}$ exactly as V would in π_{GC}, samples and sends J, performs the same checks via $\mathcal{G}_{\mathsf{trust}}$, and if the checks pass sends the same x to $\mathcal{F}_{\mathsf{GC}}$ and outputs the resulting y.

We show $H_0 \approx_c H_1$ and $H_1 \equiv H_2$. Indeed:

- **Step 1:** $H_1 \equiv H_2$**.** By construction, the joint distribution of views in H_1 and H_2 is identical: in both hybrids the verifier-side logic samples the same distribution of J, applies the same accept/abort checks using the same ideal functionalities $\mathcal{G}_{\mathsf{trust}}$, and upon acceptance sends the same x to $\mathcal{F}_{\mathsf{GC}}$ and outputs the same y returned by $\mathcal{F}_{\mathsf{GC}}$. Hence $H_1 \equiv H_2$.
- **Step 2: Bounding the distance between** H_0 **and** H_1**.** The hybrids H_0 and H_1 are identical except for the verifier output in the case where the verifier does *not* abort. Define the event

  $$\mathsf{IncorrectOutput} := \{\text{the verifier does not abort and outputs } y \neq F(x) \text{ in } H_0\}.$$

  In H_1, whenever the verifier does not abort, it outputs $y = F(x)$ by inspection of $\mathcal{F}_{\mathsf{GC}}$. Therefore, for any environment $\mathcal{Z}$,

  $$|\Pr[H_0(\kappa) = 1] - \Pr[H_1(\kappa) = 1]| \leq \Pr[\mathsf{IncorrectOutput}].$$

  It remains to upper bound $\Pr[\mathsf{IncorrectOutput}]$.
- **Step 3: Cut-and-choose soundness bound.** Let $R := [s] \setminus J$ be the unopened set, so $|R| = s/2$. Let $t := \left\lfloor \frac{|R|}{2} \right\rfloor + 1 = \left\lfloor \frac{s}{4} \right\rfloor + 1$ be the strict majority threshold among the circuits in R. Let Coll be the event that the verifier's commitment validation fails to be binding. [1] By the recompute-and-compare check of line 3, verification of opened circuits is deterministic: any opened circuit that is not a correct garbling of C under the revealed randomness, or whose labels mismatch the recorded commitments, causes the verifier to abort. Condition on $\neg\mathsf{Coll}$. If $\mathsf{IncorrectOutput}$ occurs, then the

[1] By the collision resistance of Hash (Definition 40(3)), $\Pr[\mathsf{Coll}] \leq \varepsilon_{\mathsf{cr}}(\kappa) = \mathsf{negl}(\kappa)$.

verifier did not abort and the majority decoding over the circuits in R produced an incorrect value $\hat{y} \neq F(x)$. This implies that there exist at least t indices in R whose circuits decode to $\hat{y}$ while being consistent with the posted commitments. Fix any such set $B \subseteq [s]$ of size $|B| = t$. For the verifier not to abort under $\neg\mathsf{Coll}$, none of the indices in B can be opened (otherwise an opened inconsistent/bad circuit would be detected by the checks). Thus,

$$\mathsf{IncorrectOutput} \wedge \neg\mathsf{Coll} \implies B \subseteq R.$$

Since J is uniform among all subsets of size $s/2$, for any fixed B of size t we have

$$\Pr[B \subseteq R] = \frac{\binom{s-t}{(s/2)-t}}{\binom{s}{s/2}} = \prod_{i=0}^{t-1} \frac{(s/2)-i}{s-i} \leq \left(\frac{1}{2}\right)^t.$$

Therefore,

$$\Pr[\mathsf{IncorrectOutput}] \leq \Pr[\mathsf{IncorrectOutput} \wedge \neg\mathsf{Coll}] + \Pr[\mathsf{Coll}] \leq 2^{-t} + \mathsf{negl}(\kappa) \leq 2^{-(\lfloor s/4 \rfloor + 1)} + \mathsf{negl}(\kappa).$$

In particular, choosing $s = s(\kappa)$ such that $2^{-s/4} = \mathsf{negl}(\kappa)$ makes the probability $\Pr[\mathsf{IncorrectOutput}]$ negligible.

Combining the above and using $H_1 \equiv H_2$,

$$|\Pr[H_0(\kappa) = 1] - \Pr[H_2(\kappa) = 1]| \leq 2^{-(\lfloor s/4 \rfloor + 1)} + \mathsf{negl}(\kappa),$$

so $H_0 \approx_c H_2$. Hence π_{GC} securely realizes $\mathcal{F}_{\mathsf{GC}}$.

Since $P^\star$ is statically corrupt, the adversary controls it throughout and no corruption query arises; the simulator's behaviour is unchanged. □

Corollary 6. *In the honest-evaluator, malicious-prover setting, protocol π_{GC} satisfies soundness in the sense of Definition 5: for every PPT prover $P^\star$ and every environment $\mathcal{Z}$, the output y_V of the honest evaluator V in a real execution of π_{GC} is either $\perp$ or $F(x)$, except with negligible probability.*

Proof. By Theorem 8, for every PPT adversary $\mathcal{A}$ and environment $\mathcal{Z}$, the real-world execution of π_{GC} is computationally indistinguishable from the ideal execution with $\mathcal{F}_{\mathsf{GC}}$. By Lemma 15, in the ideal world V's output is always in $\{\perp, F(x)\}$. If a PPT prover $P^\star$ could cause V to output a value $y_V \notin \{\perp, F(x)\}$ with non-negligible probability in the real world, we could use $P^\star$ and $\mathcal{Z}$ to distinguish the real and ideal executions with non-negligible advantage, contradicting Theorem 8. □

6 Conclusion

We have presented OHMG, a privacy-free garbling scheme designed for arithmetic-native circuits. The construction unifies binary, word, and one-hot wire encodings under a single global correlation Δ, and uses one-hot tensor products together with a per-arity generator $\widetilde{G}^{(I)}$ to evaluate each arithmetic gate at the cost of a single ciphertext, independently of the operand bitwidth. We proved UC authenticity of the gate protocols π_f and π_{TP}, and circuit-wide UC authenticity of π_{circ} (Theorem 6), in the GUC-RO model with a global random oracle $\mathcal{G}_{\mathsf{RO}}^X$ and a global trusted bulletin board $\mathcal{G}_{\mathsf{trust}}$, with advantage bounded by $\varepsilon_{\mathsf{hide}}(\kappa) + T_{\mathsf{hot}} \cdot q_X \cdot \mathsf{Adv}^{\Delta\mathsf{P}}(\kappa) + \mathsf{poly}(\kappa) \cdot (q_X + q_X^{\mathcal{Z}})/q$, where q_X is the number of random-oracle queries available to the adversary and $\mathsf{Adv}^{\Delta\mathsf{P}}$ is the joint Δ-power advantage of Definition 42, shown generically hard in Proposition 1. UC correctness of the end-to-end protocol π_{GC}

follows by cut-and-choose amplification, with soundness error $2^{-(\lfloor s/4 \rfloor + 1)}$ in the statistical security parameter s. The stand-alone gate protocols tolerate adaptive prover corruption (Remark 41), with the simulator equivocating its random-oracle programming against any post-message corruption request; at circuit level, the binding label commitments posted to $\mathcal{G}_{\mathsf{trust}}$ force static corruption (Remark 38). Compared to existing privacy-free garbling schemes, OHMG eliminates the bitwidth dependence inherent in Boolean decompositions, making it particularly attractive for verifying SNARK proofs and other arithmetic-heavy computations.

Several directions remain open. The constant of 1 ciphertext per arithmetic gate is tight under our current construction, but amortizing setup material across multiple gates of the same arity already saves substantial communication when circuits contain many tensor-product gates of the same shape; quantifying this saving and optimizing the per-arity generator placement is a natural next step. The cut-and-choose soundness amplification used for π_{GC} is a generic transformation; replacing it with a tighter zero-knowledge proof of correct garbling, perhaps via SNARKs or interactive oracle proofs, would reduce the prover-to-verifier communication substantially. On the implementation side, our scheme is parameterized by an elliptic curve $E(\mathbb{Z}_p)$ of prime order; while we focused on secp256k1 for concreteness, alternative curves with faster scalar multiplication or pairing-friendly structure may yield practical speedups for specific use cases.

References

[Axi24] Axiom Team. Halo2-lib bn254 pairing chip and verifier benchmarks. GitHub repository (v0.3.2), 2024. URL: https://github.com/axiom-crypto/halo2-lib.

[BA25] Osman Biçer and Ali Ajorian. AuthOr: lower cost authenticity-oriented garbling for arbitrary boolean circuits. arXiv:2501.18387 [cs.CR], 2025. URL: https://arxiv.org/abs/2501.18387.

[Bar25] Federico Barbara. Cost estimator for emulated bn254 pairings. GitHub repository, 2025. URL: https://github.com/disnocen/garbriel_bilinear.

[Bau16] Carsten Baum. On garbling schemes with and without privacy. In *Security and Cryptography for Networks (SCN 2016)*, volume 9841 of *Lecture Notes in Computer Science*, pages 468–485. Springer, Cham, 2016. DOI: 10.1007/978-3-319-44618-9_25.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *LNCS*, pages 258–275. Springer, 2005.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. Cryptology ePrint Archive, Report 2012/323, 2012. URL: https://eprint.iacr.org/2012/323.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, pages 784–796. ACM, 2012. DOI: 10.1145/2382196.2382279.

[Can20] Ran Canetti. Universally composable security. *Journal of the ACM*, 67(5):Art. 29, 94 pp. 2020. DOI: 10.1145/3402457.

[CDG+18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 280–312. Springer, 2018. DOI: 10.1007/978-3-319-78381-9_11.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography (TCC 2007)*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, Berlin, Heidelberg, 2007. DOI: 10.1007/978-3-540-70936-7_4.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. Cryptology ePrint Archive, Report 2014/553, 2014. URL: https://eprint.iacr.org/2014/553. Conference version: ACM CCS 2014, pp. 597–608, doi:10.1145/2660267.2660374.

[Con24] ConsenSys. Gnark bn254 groth16 verifier (emulated field mode). GitHub repository (v0.10.0), 2024. URL: https://github.com/ConsenSys/gnark.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003. DOI: 10.1007/978-3-540-45146-4_16.

[CS26] Céline Chevalier and Éric Sageloli. Diving deep into UC. *IACR Communications in Cryptology*, 3(1), May 4, 2026. ISSN: 3006-5496. DOI: 10.62056/a6n56chdj.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, Lecture Notes in Computer Science, pages 305–326. Springer, Berlin, Heidelberg, 2016. DOI: 10.1007/978-3-662-49896-5_11.

[Hea24] David Heath. Efficient arithmetic in garbled circuits. In *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 3–31. Springer, Cham, 2024. DOI: 10.1007/978-3-031-58740-5_1.

[HK21] David Heath and Vladimir Kolesnikov. One-hot garbling. In Giovanni Vigna and Elaine Shi, editors, *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*, pages 574–593. ACM, 2021. DOI: 10.1145/3460120.3484764.

[HMV04] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer-Verlag, New York, 2004. DOI: 10.1007/b97644.

[KMS+16] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858. IEEE, 2016. DOI: 10.1109/SP.2016.55.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: free XOR gates and applications. In *Automata, Languages and Programming (ICALP 2008), Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, Heidelberg, 2008. DOI: 10.1007/978-3-540-70583-3_40.

[Lin25] Robin Linus. BitVM3: garbled circuits for efficient computation on Bitcoin. Manuscript, 2025. URL: https://bitvm.org/bitvm3.pdf.

[LLM+24] Ryan Lavin, Xuekai Liu, Hardhik Mohanty, Logan Norman, Georges Zaarour, and Bhaskar Krishnamachari. A survey on the applications of zero-knowledge proofs. arXiv:2408.00243, 2024. URL: https://arxiv.org/abs/2408.00243.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009. DOI: 10.1007/s00145-008-9036-8.

[MW99] Ueli M. Maurer and Stefan Wolf. The relationship between breaking the Diffie–Hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997. DOI: 10.1007/3-540-69053-0_18.

[WRK18] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Optimizing authenticated garbling for faster secure two-party computation. Cryptology ePrint Archive, Report 2018/123, 2018. URL: https://eprint.iacr.org/2018/578.

[Yao86] Andrew C. Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE Computer Society Press, 1986. DOI: 10.1109/SFCS.1986.25.