

Batch IT-MAC: Program the Randomness in Succinct VOLE

Liqiang Liu, Tianren Liu

Peking University

Abstract. Many cryptographic protocols rely on affine message authentication tags of the form $\sigma = \Delta x + k$, the batch version of information-theoretic MAC. Generating the tags for a long message vector typically requires communication linear in its length. Recent chosen-input VOLE protocols achieve succinct communication, but do not allow the authenticator's key k to be fixed in advance.

We introduce *batch-MAC*, a primitive that authenticates a chosen message vector in two messages while allowing the authenticator's key vector to be generated beforehand from a short seed. For a vector of length m , we construct batch-MACs with $O(m^{2/3}\lambda)$ communication under the decisional composite residuosity (DCR) assumption and $\text{poly}(\log m, \lambda)$ communication under the learning with errors assumption. Both constructions require a common reference string (CRS).

We find applications in constrained pseudorandom functions, 2-message 2PC, and succinct zero-knowledge arguments.

1 Introduction

An Information-Theoretic Message Authentication Code (IT-MAC) can take the affine form $\sigma = x\Delta + k$, where the user holds the message x and tag σ , and the authenticator holds the keys Δ and k , satisfying $\sigma = x\Delta + k$. As long as the domain of Δ is large enough and (Δ, k) is kept from the user, the user cannot forge σ' for another message x' .

Beyond authentication, recent developments in homomorphic secret sharing (HSS) have made this affine relation extremely useful in constructing a wide range of cryptographic primitives. In particular, homomorphic evaluation on such tags and keys, sometimes combined with evaluation involving ciphertexts, enables it to be used for constructing highly efficient garbled circuits [44, 47, 49, 51, 54], trapdoor hash functions [2, 4, 30], constrained PRFs [31, 43, 55], secure multi-party computations [4, 6, 22, 34], laconic function evaluations [2, 55], etc. Moreover, it is widely used in the context of zero-knowledge proofs and signatures [5, 10–13, 29, 36, 67–69, 71, 72].

In comparison with the extended applications, the generation of MACs of this form has not been sufficiently studied. A common approach, for a vector $\mathbf{x}$ carrying a batch of messages, is to use a pseudorandom correlation generator (PCG) [17–20, 32, 48, 50, 61, 64, 70] to generate MACs for a pseudorandom vector $\mathbf{r}$, and then derandomize the MACs by publishing $\mathbf{x} - \mathbf{r}$. However, publishing

this correction requires communication linear in the message length. Viewing these correlations as Vector Oblivious Linear Evaluation (VOLE), a recent line of work [2, 4, 52] gives protocols with sublinear, and in some cases polylogarithmic, communication for a chosen $\mathbf{x}$ using simultaneous messages. However, the authenticator’s share $\mathbf{k}$ in these protocols depends on the interaction and cannot be fixed in advance. This excludes many applications. Thus, a natural question left open is:

How can we authenticate a chosen message vector using a key vector generated in advance, with minimal communication and interaction?

In the intended setting, a reusable authenticator message that lets the user evaluate $\mathbf{x}\Delta + \mathbf{k}$ for arbitrary $\mathbf{x}$ with the same fixed $\mathbf{k}$ would reveal Δ : evaluating at two vectors that differ by one in a coordinate exposes the global key. This rules out the non-interactive interface. Thus, the best we can hope for is a two-round protocol: one message from the user to the authenticator, followed by a message back from the authenticator to the user. In this work, we systematically investigate how to generate MACs for a batch of inputs $\mathbf{x}$ in this setting, with communication sublinear in $|\mathbf{x}|$.

1.1 Our Results

We formalize this task through a primitive called *Batched Message Authentication Codes*, or Batch-MAC for short. A batch-MAC is a two-message interactive protocol between an authenticator, holding a global key Δ and a one-time key vector $\mathbf{k}$, and a user, holding an input vector $\mathbf{x}$. The user derives a tag vector $\boldsymbol{\sigma}$ satisfying $\boldsymbol{\sigma} = \mathbf{x} \cdot \Delta + \mathbf{k}$.

We propose two constructions of batch-MAC with succinct communication, that is, with communication being sublinear in the message length.

Theorem 1 (DCR-based Batch-MAC). *Assuming the DCR assumption, there exists a batch-MAC scheme with $O(|\mathbf{x}|^{2/3}\lambda)$ communication.*

Theorem 2 (LWE-based Batch-MAC). *Assuming the LWE assumption, there exists a batch-MAC scheme with $\text{poly}(\log |\mathbf{x}|, \lambda)$ communication.*

We combine our batch-MAC constructions with existing techniques to obtain the following applications.

Constrained PRF with Succinct Keys. Constrained PRF, first proposed in [16, 21, 46], is a pseudorandom function with the additional ability of generating a constrained key ck_C from the master PRF key msk and a constraint C . The constrained key ck_C , together with C in the clear, can be used to compute $\text{PRF}(\text{msk}, x)$ for any input x such that $C(x) = 0$, while preserving the pseudorandomness of $\text{PRF}(\text{msk}, x)$ for x satisfying $C(x) = 1$. For the specific problem of constructing constrained PRFs for broader classes, such as NC1 or polynomial-sized circuits, a lot of work has been done [9, 14, 24–26, 28, 31, 43, 59]. However,

all of their constrained keys ck_C scale at least linearly with $|C|$ (except those employing iO). Thus, an interesting problem is to construct constrained PRFs for general circuit constraints with sublinear key sizes. A recent work [55] proposed a construction based on the DCR assumption. They achieve sublinearity through a “bundling” technique, which leads to a “bundle-size” saving in both the size and evaluation time of the master PRF key. Specifically, in their construction, they take the bundle parameter as $|C|^{1/3} \cdot \text{Depth}(C)^{-1/3}$, thus achieving $O(|C|^{2/3} \cdot \text{Depth}(C)^{1/3} \lambda)$ (master and constrained) key size.

Leveraging our batch-MAC schemes, we construct two constrained PRFs for general polynomial-sized circuits, with succinct msk and ck_C , under Paillier groups and lattices, respectively.

Theorem 3 (Constrained PRFs, Informal). *Assuming the circular power-DDH (CP-DDH) assumption in Paillier groups, there exists a (1-key, selectively secure) constrained PRF for boolean circuit C , with the master key and constrained key size both being $O(|C|^{2/3} \lambda) + \text{poly}(\lambda)$.*

Alternatively, assuming the LWE and circular power ring-LWE (CP-RLWE) assumptions, the master key size is only $\text{poly}(\lambda)$, and the constrained key size is only $\text{poly}(\log |C|, \lambda)$.

2-Message Secure 2-Party Computation. We consider the problem of constant-round secure 2-party computation (2PC). Suppose Alice inputs $\mathbf{x}$ and Bob inputs $\mathbf{y}$. After constant rounds of interaction, Alice should learn $z = C(\mathbf{x}, \mathbf{y})$ for any public circuit C . Our goal is to minimize the communication and round complexity, without employing heavy tools like FHE.

There are several primitives that imply 2-message 2PC protocols with nice communication. Laconic Function Evaluations (LFE) [60], the dual concept of FHE, can be used to construct 2-message 2PC with communication even independent of $|C|$ and $|\mathbf{x}|$. A recent line of improvements [2, 42, 60, 66] have also made the communication rate-1 to $|\mathbf{y}|$. However, they all used FHE. Recently, [22] proposed a primitive named Simultaneous Message and Succinct Secure Computation (SMS). It essentially generalizes the succinct homomorphic secret sharing (HSS) scheme in [4, 30]. The SMS protocol of [22] implies a 2-message 2PC with communication rate-1 to $|\mathbf{y}|$ and independent of $|C|$ and $|\mathbf{x}|$. However, they also used FHE. [4, 30] achieved $|\mathbf{x}|^{2/3} \lambda + |\mathbf{y}| + \text{poly}(\lambda)$ communication under mild assumptions, but only for a restricted class of functions.

Based on our batch-MAC schemes, we construct a protocol enjoying the following merits:

1. The protocol has 2 rounds of interaction: a message from Alice to Bob, followed by a message from Bob to Alice;
2. The communication is poly-logarithmic to $|\mathbf{x}|$, rate-1 to $|\mathbf{y}|$ and $|C|$;
3. Alice achieves statistical security;
4. The circuit-dependent part of Bob’s message can be computed and published offline, before the inputs are set. In this case, both Bob’s online computation and communication are independent of $|C|$.

Theorem 4 (2-Message 2PC, Informal). *Assuming the LWE and CP-RLWE assumptions, there exists a 2-message 2PC protocol for an arbitrary Boolean circuit C , with the aforementioned merits.*

We also obtain an alternative construction with larger communication, but under the Paillier groups.

Theorem 5 (2-Message 2PC under Paillier Groups, Informal). *Assuming the CP-DDH assumption in Paillier groups, there exists a 2-message 2PC protocol with communication $O(|\mathbf{x}|^{2/3}\lambda) + O(|\mathbf{y}|^{2/3}\lambda) + |\mathbf{y}| + |C| + \text{poly}(\lambda)$.*

Heuristic Circuit Argument of Knowledge. VOLE plays an important role in building efficient zero-knowledge proofs, namely the VOLE-ZKs [13, 36, 67, 69]. Typically, VOLE-ZKs enjoy highly efficient provers, and both the prover’s and verifier’s storage scales only with the memory required to verify the statement in the clear, instead of the statement size. The main drawback is that their communication and verification work typically grow linearly with the circuit size. The bottleneck is generating the VOLE. We observe that our batch-MAC schemes can be used to efficiently generate the VOLE. Specifically, by combining our LWE-based batch-MAC with QuickSilver [69], we obtain a heuristic circuit argument of knowledge with the following proposed properties:

- The total communication is polylogarithmic in the circuit size;
- The verifier can precompute circuit-dependent quantities offline before the protocol. In this case, the online verification time is almost independent of the circuit size;
- The verifier’s storage is only $\max\{O(M), \text{poly}(\log |C|, \lambda)\}$, where M denotes the working memory required to evaluate the circuit in the clear. This inherits the nice property of VOLE-ZK.

Our heuristic construction suggests a new roadmap for constructing lattice-based efficient ZKs. More details are in Section 4.

1.2 Further Related Works

Shift-Hiding PRF. A recent work [3] constructed a stronger primitive named shift-hiding PRF. Their master PRF key does not depend on the complexity of the shift function, except for its circuit depth. Their construction is based on a recently proposed assumption named decomposed-LWE. They also used LFE as a building block, which inherently needs FHE.

Garbled Input Compression. There is a similar primitive named garbled input compression, which was initiated by [8]. It aims to eliminate the $O(\lambda)$ multiplicative communication overhead of transmitting garbled input labels, when the input $\mathbf{x}$ is revealed in the clear. There is a series of follow-up works based on various assumptions [7, 35, 40, 41], among which [35] achieved $|\mathbf{x}| + \text{poly}(\lambda)$ communication for input $\mathbf{x}$ under Ring-LWE. They need an $O(|\mathbf{x}|) \text{poly}(\lambda)$ -sized

reusable non-public setup, and their hash algorithm is deterministic because privacy of $\mathbf{x}$ is not required. We note that by properly adding randomness in the hash (as shown by a recent work [56]) and leveraging the reusability of the setup, their scheme can be adapted to a batch-MAC scheme with $\sqrt{|\mathbf{x}|} \text{poly}(\lambda)$ communication. In the other direction, our results imply garbled input compression schemes (for labels sharing global Δ , which they named weak batch select scheme), with $O(|\mathbf{x}|^{2/3} \lambda)$ communication under DCR, or $\text{poly}(\log |\mathbf{x}|, \lambda)$ communication under LWE.

Concurrent Work. In a concurrent and independent paper [56], the authors considered a question similar to our batch-MAC, which they called “chosen-input VOLE in the staged mode”. Their construction follows from compressing the linear-sized setup message in the garbled input compression scheme of [35]. When viewed as a batch-MAC, they achieve $\text{poly}(\log |\mathbf{x}|, \lambda)$ total communication, assuming Ring-LWE, in the programmable random-oracle model. In contrast, we achieve the same asymptotic cost assuming only LWE, and we additionally show a DCR-based construction. As for ZK, they show a complete protocol with malicious security and concrete efficiency. We only show a heuristic result as inspiration for future work. However, our protocol additionally enjoys an efficient online phase and small verifier storage.

1.3 Technical Overview

Group-based Construction This section uses the following notations. For any vector $\mathbf{v}$ and group element g , let $g^{\mathbf{v}} = (g^{v_1}, g^{v_2}, \dots)$ denote the point-wise exponent. For any vector $\mathbf{a}, \mathbf{b}$, let $\mathbf{a} \otimes \mathbf{b} = (a_1 b_1, a_2 b_2, \dots)$ denote the point-wise product.

Our construction is inspired by the recent group-based half-chosen VOLE [4]. First, briefly recap their construction. In their setting, the user has a vector $\mathbf{x}$, the authenticator has a scalar y , and they jointly get additive shares of $y\mathbf{x}$ after a single round of simultaneous messages. CRS is generated in the setup phase and consists of a DDH-hard group $(\mathbb{G}, g)$ and $m^{\frac{1}{3}}$ group elements $g^{\mathbf{k}} = (g_1^k, \dots, g_{m^{1/3}}^k)$. Interpret the input $\mathbf{x}$ as a $m^{\frac{2}{3}} \times m^{\frac{1}{3}}$ matrix $\mathbf{X}$. The user sends the hash value $g^{\mathbf{Xk}}$. The authenticator samples random $\mathbf{t}$ and sends $g^{\mathbf{kt}^\top + yI}$. The user and the authenticator can then compute

$$g^{\mathbf{Xkt}^\top + y\mathbf{X}} \text{ (from } g^{\mathbf{kt}^\top + yI}, \mathbf{X}) \text{ and } g^{\mathbf{Xkt}^\top} \text{ (from } g^{\mathbf{Xk}}, \mathbf{t})$$

respectively. They are multiplicative shares of $g^{y\mathbf{X}}$ and can be converted into additive shares of $y\mathbf{X}$ via distributed discrete logarithm (DDLog). DDLog is an efficient algorithm such that $\text{DDLog}(g^{r+x}) - \text{DDLog}(g^r) = x$ with good probability when x is small.

To relax the size restriction, [57] invents DDLog on the Paillier group $(\mathbb{G}, g, f)$. The new algorithm satisfies $\text{DDLog}(g^r f^x) - \text{DDLog}(g^r) = x \pmod{N}$ where f is an order- N ($\approx 2^\lambda$) group element.

To program the authenticator-side randomness, our construction has 3 steps:

1. *Generate additive shares of $\mathbf{X}\mathbf{w}$.* This step is similar to [4]. The public key contains a Paillier group $\mathbf{pp} = (\hat{\mathbb{G}}, \hat{g}, \hat{f})$ and a length- $m^{1/3}$ vector $\hat{g}^{\mathbf{k}}$. The user sends the hash vector $\hat{g}^{\mathbf{Xk}}$. The authenticator answers $\hat{g}^{s\mathbf{k}} \odot \hat{f}^{\mathbf{w}}$. The user and the authenticator can compute

$$\hat{g}^{s\mathbf{Xk}} \odot \hat{f}^{\mathbf{Xw}} \text{ and } \hat{g}^{s\mathbf{Xk}}$$

respectively. They are the multiplicative shares of $\hat{f}^{\mathbf{Xw}}$, and can be distributively converted into additive shares of $\mathbf{Xw}$ using DDLog.

2. *Derandomize authenticator's share.* The authenticator also sends its share plus $\mathbf{r}$. Now the user and the authenticator can compute

$$\mathbf{r} - \mathbf{Xw} \text{ and } \mathbf{r} \quad \text{respectively.}$$

3. The public key also contains another Paillier group $\mathbf{pp} = (\mathbb{G}, g, f)$. The authenticator sends a random length- $m^{1/3}$ vector $\hat{g}^{\mathbf{t}}$, allowing the user to compute

$$g^{\mathbf{rt}^\top - \mathbf{Xwt}^\top}.$$

The authenticator sends $g^{\mathbf{wt}^\top} \odot f^{y\mathbf{I}}$, allowing the user to compute $g^{\mathbf{Xwt}^\top} \odot f^{y\mathbf{X}}$. The user and the authenticator can compute

$$g^{\mathbf{rt}^\top} \odot f^{y\mathbf{X}} \text{ and } g^{\mathbf{rt}^\top}$$

respectively. They are the multiplicative shares of $\hat{f}^{y\mathbf{X}}$, and can be distributively converted into additive shares of $y\mathbf{X}$ using DDLog.

Regrouping all the actions, they fit into the desired interaction pattern.

Setup Sample two Paillier groups $\mathbf{pp} = (\mathbb{G}, g, f)$, $\hat{\mathbf{pp}} = (\hat{\mathbb{G}}, \hat{g}, \hat{f})$. Sample $\hat{g}^{\mathbf{k}}$.

KeyGen (Authenticator) Sample random $\mathbf{r}, \mathbf{t}$. Output $\text{DDLog}(\mathbf{pp}, g^{\mathbf{rt}^\top})$.

Hash (User) View the input as a $m^{\frac{2}{3}} \times m^{\frac{1}{3}}$ matrix $\mathbf{X}$. Compute the hash value $\hat{g}^{\mathbf{Xk}}$ from $\mathbf{X}, \hat{g}^{\mathbf{k}}$.

Auth (Authenticator) Sample random $s, \mathbf{w}$.

Send $\hat{g}^{s\mathbf{k}} \odot \hat{f}^{\mathbf{w}}, \text{DDLog}(\hat{\mathbf{pp}}, \hat{g}^{s\mathbf{Xk}}) + \mathbf{r}, g^{\mathbf{t}}, g^{\mathbf{wt}^\top} \odot f^{y\mathbf{I}}$.

Ext (User) Compute $\text{DDLog}(\hat{\mathbf{pp}}, \hat{g}^{s\mathbf{Xk}} \odot \hat{f}^{\mathbf{Xw}})$ from $\mathbf{X}, \hat{g}^{s\mathbf{k}} \odot \hat{f}^{\mathbf{w}}$.

Compute $\mathbf{r} - \mathbf{Xw}$ from $\text{DDLog}(\hat{\mathbf{pp}}, \hat{g}^{s\mathbf{Xk}} \odot \hat{f}^{\mathbf{Xw}}), \text{DDLog}(\hat{\mathbf{pp}}, \hat{g}^{s\mathbf{Xk}}) + \mathbf{r}$.

Compute $g^{\mathbf{rt}^\top - \mathbf{Xwt}^\top}$ from $\mathbf{r} - \mathbf{Xw}, g^{\mathbf{t}}$.

Compute $g^{\mathbf{Xwt}^\top} \odot f^{y\mathbf{X}}$ from $\mathbf{X}, g^{\mathbf{wt}^\top} \odot f^{y\mathbf{I}}$.

Multiple the last two, get $g^{\mathbf{rt}^\top} \odot f^{y\mathbf{X}}$, and output $\text{DDLog}(\mathbf{pp}, g^{\mathbf{rt}^\top} \odot f^{y\mathbf{X}})$.

Lattice-based Construction Our construction is built on existing lattice-based succinct heVOLE [2]. Their protocol, ignoring many details, works as follows. There are two parties, sender (= our user) and receiver (= our authenticator). The sender is given input $\mathbf{x}$ and generates a digest $\mathbf{d}$. The receiver is given

input $\mathbf{y}$, samples a secret key $\mathbf{s}_0$, and generates an encoding. After exchanging the digest and encoding, they jointly get approximated additive shares of $\mathbf{x} \oplus \mathbf{y}$,

$$\text{Sender's share} - \underbrace{\mathbf{P}(\mathbf{d} \otimes \mathbf{s}_0)}_{\text{Receiver's share}} \approx \frac{q}{p}(\mathbf{x} \otimes \mathbf{y}).$$

Here $\mathbf{P}$ is a public low-norm matrix determined by the CRS. The error can be removed w.h.p. by rounding.

To derandomize the share of the receiver (=authenticator), let it sample a short seed $\mathbf{s}$ and additionally send

$$\mathbf{A}\mathbf{s} + \text{err} - \mathbf{d} \otimes \mathbf{s}_0$$

to the sender (=user). Then

$$\underbrace{\text{Sender's share} + \mathbf{P}(\mathbf{A}\mathbf{s} + \text{err} + \mathbf{d} \otimes \mathbf{s}_0)}_{\text{User's share}} - \underbrace{\mathbf{P}\mathbf{A}\mathbf{s}}_{\text{Authenticator's share}} \approx \frac{q}{p}(\mathbf{x} \otimes \mathbf{y}).$$

Recap the Lattice-based hcVOLE of [2]. In the rest of the section, we show how [2] satisfies the claimed property. Their protocol is built upon [4] whose communication is $m^{2/3} \cdot \text{poly}(\lambda)$. They further reduce the communication to $\text{poly}(\log m, \lambda)$ by leveraging the inherent recursive structure of the receiver's share. Sample the public matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ for $n < m$. The sender computes its hash digest as $\mathbf{d} := \mathbf{A} \cdot \mathbf{x}$. The receiver computes an LWE ciphertext for y as $\mathbf{C} := \mathbf{A}^\top \cdot \mathbf{S} + \mathbf{E} + \mathbf{I}_m \cdot y$, where $\mathbf{S} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ is the LWE secret and $\mathbf{E}$ is the error. Then the sender computes

$$\mathbf{w} := \mathbf{x}^\top \cdot \mathbf{C} = \mathbf{x}^\top \cdot \mathbf{A}^\top \cdot \mathbf{S} + \mathbf{x}^\top \cdot \mathbf{E} + \mathbf{x}^\top \cdot y,$$

and the receiver computes

$$\mathbf{v} := \mathbf{d}^\top \cdot \mathbf{S} = \mathbf{x}^\top \cdot \mathbf{A}^\top \cdot \mathbf{S}.$$

Now $\mathbf{w} \approx \mathbf{v} + \mathbf{x}^\top \cdot y$, assuming $\|\mathbf{x}\|_\infty$ is small.

The key observation in [2] is that $\mathbf{v}$ is a linear function of $\mathbf{d} \otimes \text{vec}(\mathbf{S})$, where vec is the vectorisation function. Moreover, $\mathbf{d}$ is known by the sender, and $\text{vec}(\mathbf{S})$ is known by the receiver, and the tensor product is simply a repetition of VOLEs. Given that $|\mathbf{d}| < |\mathbf{x}|$, if they could get shares of $\mathbf{d} \otimes \text{vec}(\mathbf{S})$ by a recursive invocation of hcVOLE for shorter inputs, the total communication would be reduced. However, $\text{vec}(\mathbf{S})$ is too long; thus, they instead used another LWE instance to pseudorandomly generate $\mathbf{S}$.

To be more precise, they directly constructed a non-interactive tensor product scheme (which they named oblivious tensor evaluation, or OTE for short). Let the receiver's input be $\mathbf{y}$ of length $k \in \Theta(\lambda)$. Now set the LWE secret key $\mathbf{S}$ to be generated by $m \cdot k$ LWE instances, using the same secret vector $\mathbf{s} \in \mathbb{Z}_q^k$ but different public matrices $\mathbf{B}_1, \dots, \mathbf{B}_{m \cdot k}$, or rather,

$$\mathbf{S} = (\mathbf{B}_1 \cdots \mathbf{B}_{m \cdot k}) \cdot (\mathbf{I}_{m \cdot k} \otimes \mathbf{s}) + \mathbf{E}' = \mathbf{B} \cdot (\mathbf{I}_{m \cdot k} \otimes \mathbf{s}) + \mathbf{E}'.$$

In this case, the receiver's message will be formulated as

$$\mathbf{C} = \mathbf{A}^\top \cdot (\mathbf{B} \cdot (\mathbf{I}_{m \cdot k} \otimes \mathbf{s}) + \mathbf{E}') + \mathbf{E} + \mathbf{I}_m \otimes \mathbf{y}^\top$$

Thus the sender gets

$$\begin{aligned} \mathbf{w} &:= \mathbf{x}^\top \cdot \mathbf{C} \approx \mathbf{x}^\top \mathbf{A}^\top \cdot \mathbf{B} (\mathbf{I}_{m \cdot k} \otimes \mathbf{s}) + \mathbf{x}^\top \otimes \mathbf{y}^\top \\ &= \mathbf{d}^\top \cdot \mathbf{B} (\mathbf{I}_{m \cdot k} \otimes \mathbf{s}) + \mathbf{x}^\top \otimes \mathbf{y}^\top, \end{aligned}$$

assuming that $\|\mathbf{A}\|_\infty$ and $\|\mathbf{x}\|_\infty$ are small. This can be reduced to a strictly smaller OTE scheme for $\mathbf{d}$ and $\mathbf{s}$. However, the size of $\mathbf{C}$ is still too big. This can again be solved by the balancing trick: instead of hashing the whole vector $\mathbf{x}$, divide it into short sub-vectors and run the aforementioned tensor product scheme on each of them. Now we use M to denote the length of the sender's input, and $M \gg m$. Each iteration of the aforementioned OTE scheme shortens the sender's input length by $\frac{m}{n}$. The sender only needs to send the digest of the final iteration, while the receiver needs to send a ciphertext for each iteration. This gives a non-interactive OTE scheme with $\text{poly}(\log M, \lambda)$ communication.

At each level of the iteration, the receiver's share $\mathbf{w}_i$ is a public linear function of $\mathbf{d}_i \otimes \mathbf{s}_i$, in which $\mathbf{d}_i$ and $\mathbf{s}_i$ are the digest and secret key for that level. Thus, assuming that $\mathbf{s}$ and $\mathbf{d}$ are the secret key and digest for the final level, the receiver's output for $\mathbf{x} \otimes \mathbf{y}$ will be $\mathbf{v} := \mathbf{P} \cdot (\mathbf{d} \otimes \mathbf{s})$, where $\mathbf{P}$ can be publicly derived from the CRS.

1.4 Overview: Applications of Batch-MAC

Recent progress in secret sharing and MPC has leveraged MACs in the form of $\sigma_{\mathbf{x}} = \mathbf{x} \cdot \Delta + k_{\mathbf{x}}$ to construct many important primitives, including constrained PRFs, garbled circuits, trapdoor hash functions, etc. At a high level, the core technique is that the user and the authenticator can locally apply homomorphic evaluations on the tags and keys, respectively. Our batch-MAC scheme, in turn, resolves the problem of generating the MACs for $\mathbf{x}$ with succinct communication. Therefore, by incorporating our batch-MAC into those applications, we acquire straightforward improvements.

Constrained PRF. In a constrained PRF (CPRF), an algorithm `Constrain` takes the master PRF key msk and a constraint circuit C as inputs and outputs a constrained key ck_C . The two evaluation algorithms `Eval` and `CEval`, with msk and (C, ck_C) respectively, should evaluate (on x) to equal values if $C(x) = 0$. Meanwhile, $\text{Eval}(\text{msk}, x)$ should remain pseudorandom if $C(x) = 1$, even given ck_C . Simialr to former works [14, 31, 43, 55], our constrained PRF follows the CMPR23 blueprint [31]. The high-level idea is to let the two parties, each holding msk and (C, ck_C) , establish additive shares of $C(x) \cdot F(k, x)$ for a PRF F .

We borrow two techniques from previous works:

- The algebraic homomorphic MAC (aHMAC) scheme [43] states that the authenticator can derive an evaluation key evk from the (global) MAC key Δ . Further, there are two evaluation algorithms, EvalTag and EvalKey , satisfying

$$\begin{aligned}\sigma_z &\leftarrow \text{EvalTag}(\text{evk}, G, \mathbf{x}, \sigma_{\mathbf{x}}), \quad k_z \leftarrow \text{EvalKey}(\Delta, G, k_{\mathbf{x}}), \\ \sigma_z &= k_z + z \cdot \Delta, \quad \text{for } z = G(\mathbf{x}), \quad G \text{ is an arbitrary circuit.}\end{aligned}$$

- The extended HSS evaluation algorithm [31] states that two parties, starting from additive shares of $z \cdot \Delta$ and encryptions of $\mathbf{y}$ under Δ (named $\text{ct}_{\Delta}(\mathbf{y})$), can locally derive additive shares of $z \cdot P(\mathbf{y})$ for any NC1 circuit P .

Our CPRF generally works as follows: first, invoke batch-MAC to establish $\sigma_C = k_C + C \cdot \Delta$; second, invoke aHMAC to turn them into $\sigma_{C(x)} = k_{C(x)} + C(x) \cdot \Delta$; then, view the MAC as additive shares of $C(x) \cdot \Delta$ and turn it into shares of $C(x) \cdot F(\mathbf{k}, x)$ with the aid of $\text{ct}_{\Delta}(\mathbf{k})$. More formally, we first sample a global key Δ and compute the MAC key for C using randomness $\mathbf{r}_{\text{key}}$:

$$k_C \xleftarrow{\$} \text{batchMAC.KeyGen}(1^\lambda; \mathbf{r}_{\text{key}}).$$

Note that k_C and the randomness $\mathbf{r}_{\text{key}}$ is independent of C . Next, sample a PRF key $\mathbf{k}$ for an NC1 PRF F , and compute its encryption under Δ : $\text{ct}_{\Delta}(\mathbf{k})$. The PRF master key is

$$\text{msk} := (\Delta, \mathbf{r}_{\text{key}}, \text{ct}_{\Delta}(\mathbf{k})).$$

When C is given, the Constrain algorithm runs as follows: compute

$$\begin{aligned}d &\xleftarrow{\$} \text{batchMAC.Hash}(\text{crs}, C; \mathbf{r}_{\text{tag}}), \\ e &\leftarrow \text{batchMAC.Auth}(\text{crs}, \Delta, d, \mathbf{r}_{\text{key}}),\end{aligned}$$

and compute the aHMAC evk out of Δ . The constrained key ck_C is

$$\text{ck}_C := (\mathbf{r}_{\text{tag}}, e, \text{evk}, \text{ct}_{\Delta}(\mathbf{k})).$$

The evaluation algorithm at x goes as follows:

$$\begin{aligned}\text{Eval} : \quad &k_C \xleftarrow{\$} \text{batchMAC.KeyGen}(1^\lambda; \mathbf{r}_{\text{key}}); \quad k_{C(x)} \leftarrow \text{EvalKey}(\Delta, U_x(\cdot), k_C). \\ \text{CEval} : \quad &\sigma_C \leftarrow \text{batchMAC.Ext}(C, e, \mathbf{r}_{\text{tag}}); \quad \sigma_{C(x)} \leftarrow \text{EvalTag}(\text{evk}, U_x(\cdot), C, \sigma_C).\end{aligned}$$

Define $U_x(C) = C(x)$ to be a universal function, and $F_x(\mathbf{k}) = F(\mathbf{k}, x)$. Then,

$$\begin{aligned}\text{Eval} : \quad &(\text{share of } C(x) \cdot F(\mathbf{k}, x)) \leftarrow \text{ExtHssEval}(k_{C(x)}, \text{ct}_{\Delta}(\mathbf{k}), F_x(\cdot)). \\ \text{CEval} : \quad &(\text{share of } C(x) \cdot F(\mathbf{k}, x)) \leftarrow \text{ExtHssEval}(\sigma_{C(x)}, \text{ct}_{\Delta}(\mathbf{k}), F_x(\cdot)).\end{aligned}$$

The sizes of Δ , $\text{ct}_{\Delta}(\mathbf{k})$, and evk are all $\text{poly}(\lambda)$. When the batch-MAC is instantiated under DCR, $|\mathbf{r}_{\text{key}}|$, $|\mathbf{r}_{\text{tag}}|$, and $|e|$ are all $O(|C|^{2/3}\lambda)$. We have $|\text{msk}| = O(|C|^{2/3}\lambda) + \text{poly}(\lambda)$ and $|\text{ck}_C| = O(|C|^{2/3}\lambda) + \text{poly}(\lambda)$. When the batch-MAC is instantiated under LWE, $|\mathbf{r}_{\text{key}}|$ and $|\mathbf{r}_{\text{tag}}|$ are $\text{poly}(\lambda)$, and $|e|$ is $\text{poly}(\log |C|, \lambda)$. Thus, we have $|\text{msk}| = \text{poly}(\lambda)$ and $|\text{ck}_C| = \text{poly}(\log |C|, \lambda)$.

Succinct 2-Party Computation. Garbled circuits (GC), as one of the most significant 2PC schemes, enjoy constant rounds of communication. In a standard boolean garbling scheme, before the inputs are given, the garbler generates labels (L_0^w, L_1^w) for each input wire w of the circuit, and computes a garbled version of the circuit, namely $\hat{C} \leftarrow \text{Gb}(\text{sk}, C)$. Later, assuming that the evaluator obtains labels $L_{\mathbf{x}} := (L_{x_1}^{w_1}, \dots, L_{x_m}^{w_m})$ for inputs $\mathbf{x} = (x_1, \dots, x_m)$, it can compute $z \leftarrow \text{Ev}(\hat{C}, L_{\mathbf{x}})$ which equals to $C(\mathbf{x})$.

In almost all modern garbling schemes, the input labels $L_{\mathbf{x}}$ inherits the form of $L_{\bar{\mathbf{x}}} = \bar{\mathbf{x}} \cdot \Delta + K_{\bar{\mathbf{x}}}$ for global key Δ , where $K_{\bar{\mathbf{x}}}$ is the wire-key sampled by the garbler. $\bar{\mathbf{x}} := \mathbf{x} \oplus \mathbf{r}$ is the masked wire value known by the evaluator, with $\mathbf{r}$ known by the garbler. Our key observation is that this type of label can be transmitted to the evaluator with succinct communication, through our batch-MAC schemes. In particular, we combine our batch-MAC scheme with the ILL25 1-bit-per-gate garbling scheme [44]¹. To summarize, in their construction,

$$\begin{aligned} \text{Garbler} : (\Delta, \mathbf{r}, K_{\bar{\mathbf{x}}} \leftarrow \$) &\xrightarrow{\text{Gb}} \hat{C}, \quad |\hat{C}| = |C| + \text{poly}(\lambda). \\ \text{Evaluator} : (\bar{\mathbf{x}} := \mathbf{x} \oplus \mathbf{r}, L_{\bar{\mathbf{x}}} = K_{\bar{\mathbf{x}}} + \Delta \cdot \bar{\mathbf{x}}, \hat{C}) &\xrightarrow{\text{Ev}} z. \end{aligned}$$

We consider the secure 2-party computation problem where Alice has input $\mathbf{x} \in \{0, 1\}^m$ and Bob has input $\mathbf{y} \in \{0, 1\}^\ell$. After a message from Alice to Bob and a subsequent message from Bob to Alice, Alice acquires $C(\mathbf{x}, \mathbf{y})$ for an arbitrary public circuit C .

1. Let Bob act as the garbler. It masks $\mathbf{y}$ as $\bar{\mathbf{y}} := \mathbf{y} \oplus \mathbf{r}$. It next invokes `batchMAC.KeyGen` to generate $K_{\mathbf{x}}$ and $K_{\bar{\mathbf{y}}}$. It then samples Δ and generates the garbled circuit:

$$\text{Bob} : (\Delta, \mathbf{0}, \mathbf{r}, K_{\mathbf{x}}, K_{\bar{\mathbf{y}}}) \xrightarrow{\text{Gb}} \hat{C}.$$

These can be done even before $\mathbf{x}$ and $\mathbf{y}$ are set.

2. Let Alice act as the evaluator. It invokes `batchMAC.Hash` on $\mathbf{x}$ and sends the output $d_{\mathbf{x}}$ to Bob.
3. Upon receiving $d_{\mathbf{x}}$ from Alice, Bob computes

$$\begin{aligned} \text{Bob} : d_{\bar{\mathbf{y}}} &\leftarrow \text{batchMAC.Hash}(\bar{\mathbf{y}}), \\ e_{\mathbf{x}} &\leftarrow \text{batchMAC.Auth}(\Delta, d_{\mathbf{x}}, K_{\mathbf{x}}), \quad e_{\bar{\mathbf{y}}} \leftarrow \text{batchMAC.Auth}(\Delta, d_{\bar{\mathbf{y}}}, K_{\bar{\mathbf{y}}}). \end{aligned}$$

Bob then sends to Alice

$$\text{Bob} \rightarrow \text{Alice} : (\bar{\mathbf{y}}, \hat{C}, e_{\mathbf{x}}, e_{\bar{\mathbf{y}}}).$$

4. Upon receiving the message from Bob, Alice (as the evaluator) computes

$$\begin{aligned} \text{Alice} : L_{\mathbf{x}} &= K_{\mathbf{x}} + \mathbf{x} \cdot \Delta \leftarrow \text{batchMAC.Ext}(\mathbf{x}, e_{\mathbf{x}}), \\ L_{\bar{\mathbf{y}}} &= K_{\bar{\mathbf{y}}} + \bar{\mathbf{y}} \cdot \Delta \leftarrow \text{batchMAC.Ext}(\bar{\mathbf{y}}, e_{\bar{\mathbf{y}}}), \\ (\mathbf{x}, \bar{\mathbf{y}}, L_{\mathbf{x}}, L_{\bar{\mathbf{y}}}, \hat{C}) &\xrightarrow{\text{Ev}} z. \end{aligned}$$

¹ There are also other 1-bit-per-gate GCs [51, 54]. The forms of their labels are almost identical.

Note that $|\mathbf{y}| = \ell$ and $|\hat{C}| = |C| + \text{poly}(\lambda)$. For the DCR-based instantiation of batch-MAC, $|d_{\mathbf{x}}| = O(m^{2/3}\lambda)$, $|e_{\mathbf{x}}| = O(m^{2/3}\lambda)$, and $|e_{\mathbf{y}}| = O(\ell^{2/3}\lambda)$. For the LWE-based instantiation of batch-MAC, $|d_{\mathbf{x}}| = \text{poly}(\log m, \lambda)$, $|e_{\mathbf{x}}| = \text{poly}(\log m, \lambda)$, and $|e_{\mathbf{y}}| = \text{poly}(\log \ell, \lambda)$.

2 Notations and Preliminary

Notations. For any $n \in \mathbb{N}$, we define $[n] := \{1, \dots, n\}$ and $[0..n] := \{0, 1, \dots, n\}$. In most cases, we use bold lowercase letters to denote vectors, and bold uppercase letters to denote matrices. Let $\|\mathbf{x}\|_{\infty}$ be the infinity norm of vector $\mathbf{x}$. By default, all vectors are columns, with the exception of the gadget vector $\mathbf{g}_q = (1, 2, \dots, 2^{\log q})$. In this paper, a vector of length n is indexed by $1, 2, \dots, n$. Tensor product is defined as $(a_{ij})_{i,j} \otimes \mathbf{B} = (a_{ij} \cdot \mathbf{B})_{i,j}$. We use $\odot$ to denote the pointwise product, i.e. $\mathbf{x} \odot \mathbf{y} = (\mathbf{x}[1]\mathbf{y}[1], \dots, \mathbf{x}[n]\mathbf{y}[n])$.

Given two integers p and x , we use $\lfloor x \rfloor_p$ to denote the integer y such that $y \cdot p$ is the multiple of p that is closest to x . For a vector $\mathbf{x}$, let $\lfloor \mathbf{x} \rfloor_p$ denote $\lfloor \cdot \rfloor_p$ of each entry of $\mathbf{x}$.

For a generator g of group G and a vector $\mathbf{x} = (x_1, \dots, x_n)$, we use $g^{\mathbf{x}}$ to denote the vector $(g^{x_1}, \dots, g^{x_n})$.

Paillier Groups and the DCR Assumption. We now recall the decisional composite residuosity (DCR) assumption over the Paillier group [33, 58]. We recall that the Paillier group is $\mathbb{Z}_{N^2}^*$, where N is the product of two random safe primes. The DCR assumption states that over the Paillier group (where the factorisation of N is kept secret), it is hard to distinguish between a random N -th power and a random element of the group.

Definition 1 (Decisional Composite Residuosity). *We say that the decisional composite residuosity (DCR) assumption holds over the Paillier group, if the following distributions are computationally indistinguishable*

$$\left\{ g, N \left| \begin{array}{l} p, q \xleftarrow{\$} \text{random } \lambda\text{-bit safe-primes} \\ N \leftarrow p \cdot q \\ g \xleftarrow{\$} \mathbb{Z}_{N^2}^* \end{array} \right. \right\}$$

$$\left\{ g^N, N \left| \begin{array}{l} p, q \xleftarrow{\$} \text{random } \lambda\text{-bit safe-primes} \\ N \leftarrow p \cdot q \\ g \xleftarrow{\$} \mathbb{Z}_{N^2}^* \end{array} \right. \right\}$$

To unify different instantiations of group-based distributed discrete logarithm, [1] invented the non-interactive discrete logarithm sharing (NIDLS) framework. Our group-based batch-MAC is built upon the Paillier group instantiation of the NIDLS framework. We summarize it below.

Lemma 1 (The Paillier Instantiation [1, 57, 63]). *There exists a triple of efficient algorithms $(\text{Gen}_{\text{Paillier}}, \mathcal{D}, \text{DDLog})$ with the following syntax:*

- $\text{Gen}_{\text{Paillier}}(1^\lambda)$ outputs a tuple of public parameters $\text{pp} := (G, F, H, f, N, \ell)$ where
 - G is a finite cyclic group consisting of squares of $\mathbb{Z}_{N^2}^*$. It is of order $N \cdot \Phi(N)/4$. N is the product of two λ -bit safe-primes;
 - F and H are subgroups of G such that $G = F \times H$;
 - $F = \langle f \rangle$, $f = (1 + N)$ and $|F| = N$. The discrete logarithm problem is easy in F ;
 - H is the subgroup of $2N$ -th powers of $\mathbb{Z}_{N^2}^*$;
 - ℓ is a positive integer, specifically $\ell := 2^\lambda \cdot N^2$.
- $\mathcal{D}(1^\lambda, \text{pp})$ outputs a random element $g \in G$ along with auxiliary information ρ . It is instantiated by sampling $\rho \xleftarrow{\$} \mathbb{Z}_{N^2}^*$ and outputting $(g := \rho^2, \rho)$. With overwhelming probability, g is a generator of G .
- $\text{DDLog}(\text{pp}, g)$ is deterministic and outputs an element $s \in \mathbb{Z}_N$.

Additionally, the following properties hold:

- For every PPT adversary $\mathcal{A}$,

$$\Pr \left[s_0 - s_1 \neq m \bmod N \mid \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ (g_0, m) \xleftarrow{\$} \mathcal{A}(1^\lambda, \text{pp}) \\ g_1 \leftarrow f^m \cdot g_0 \\ s_0 \leftarrow \text{DDLog}(\text{pp}, g_0) \\ s_1 \leftarrow \text{DDLog}(\text{pp}, g_1) \end{array} \right] \leq \text{negl}(\lambda).$$

- The following distributions are statistically indistinguishable

$$\left\{ \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \text{pp}, g, \rho, g^r \mid \begin{array}{l} (g, \rho) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ r \xleftarrow{\$} [\ell] \end{array} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \text{pp}, g, \rho, h \mid \begin{array}{l} (g, \rho) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ h \xleftarrow{\$} \langle g \rangle \end{array} \end{array} \right\}.$$

The security of our batch-MAC scheme relies on the following n -ary enhanced DDH assumption, which is further implied by the DCR assumption [4]. It states that given the parameters of the Paillier group and $n+1$ group elements $g_0, \dots, g_n$ sampled from $\mathcal{D}$ (along with the corresponding auxiliary information $\rho_0, \dots, \rho_n$), it is hard to distinguish between $(g_0^w, \dots, g_n^w)$ for a random w and $(f^{r_0} \cdot g_0^w, \dots, f^{r_n} \cdot g_n^w)$ for random $r_0, \dots, r_n \in \mathbb{Z}_N$.

Definition 2 (The n -ary Enhanced DDH Assumption). We say that the n -ary Enhanced DDH assumption holds in the Paillier group if the following distributions are computationally indistinguishable

$$\left\{ \begin{array}{l} \text{pp} \\ g_0, \dots, g_n \\ \rho_0, \dots, \rho_n \\ g_0^w, \dots, g_n^w \end{array} \mid \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [0..n] : (g_j, \rho_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ w \xleftarrow{\$} [\ell] \end{array} \right\},$$

$$\left\{ \begin{array}{l|l} \text{pp} & \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ g_0, \dots, g_n & \forall j \in [0..n] : (g_j, \rho_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ \rho_0, \dots, \rho_n & w \xleftarrow{\$} [\ell] \\ f^{r_0} \cdot g_0^w, \dots, f^{r_n} \cdot g_n^w & \forall j \in [0..n] : r_j \xleftarrow{\$} \mathbb{Z}_N \end{array} \right\}.$$

Lemma 2 (DCR Implies Enhanced DDH). *The DCR assumption implies the n -ary enhanced DDH assumption for any polynomial $n(\lambda)$.*

The Learning with Errors Assumption. We say that a low-norm distribution $\chi(1^\lambda)$ over $\mathbb{Z}$ is $B(\lambda)$ -bounded if: $\Pr[\|e\|_\infty \leq B(\lambda) \mid e \xleftarrow{\$} \chi(1^\lambda)] = 1$. Sometimes we abuse the notation as $\mathbf{v} \xleftarrow{\$} \chi(1^\lambda)$, meaning that each entry of $\mathbf{v}$ is independently sampled from $\chi(1^\lambda)$.

In this work, we assume the learning with errors (LWE) assumption introduced by Regev [62].

Assumption 6 (Learning With Errors) *Let $k := k(\lambda)$, $m := m(\lambda)$ and $q := q(\lambda)$ be positive integers. Let $\chi(1^\lambda)$ be a low-norm distribution over $\mathbb{Z}$. The (χ, k, m, q) -LWE assumption states that, for every PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that, for every $\lambda \in \mathbb{N}$, we have*

$$\left| \Pr \left[\mathcal{A}(1^\lambda, \mathbf{A}, \mathbf{b}) = 1 \mid \begin{array}{l} \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times k}, \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k, \mathbf{e} \xleftarrow{\$} \chi^m(1^\lambda) \\ \mathbf{b} \leftarrow \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \end{array} \right] \right. \\ \left. - \Pr \left[\mathcal{A}(1^\lambda, \mathbf{A}, \mathbf{b}) = 1 \mid \begin{array}{l} \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times k} \\ \mathbf{b} \xleftarrow{\$} \mathbb{Z}_q^m \end{array} \right] \right| \leq \text{negl}(\lambda).$$

As in the previous work [2], we also assume the non-uniform LWE (NLWE). It is a variant of standard LWE, with the difference that the matrix $\mathbf{A}$ is no longer uniformly random over $\mathbb{Z}_q^{m \times k}$, but over $\mathbb{Z}_2^{m \times k}$. [15] has shown that non-uniform LWE is at least as hard as LWE, with a slightly increased parameter size.

Lemma 3 (Hardness of NLWE). *Assume the hardness of (χ, k, m, q) -LWE. Then, $(\chi, k \cdot \log q, m, q)$ -NLWE is hard.*

3 Batched Message Authentication Codes

Definition 3 (batch-MAC). *Let $m = m(\lambda)$ and $p = p(\lambda)$ be positive integers, and $\mathcal{R}$ be a ring. A batched MAC (batch-MAC) scheme for message space $\mathcal{R}^m$ and secret key space $\mathcal{R}$ consists of a tuple of efficient algorithms (Setup, KeyGen, Hash, Auth, Ext) with the following syntax:*

- **Setup**(1^λ): *The setup algorithm is probabilistic, takes the security parameter 1^λ as input, and outputs a public key pk .*
- **KeyGen**(pk): *The receiver's evaluation algorithm is probabilistic, takes the public key pk as input, and outputs the (one-time) MAC key $\mathbf{k}$ and the authenticator's private state st_A .*

- $\text{Hash}(\text{pk}, \mathbf{x})$: The hashing algorithm is probabilistic, takes as input pk and the user's message $\mathbf{x} \in \mathcal{R}^m$, outputs a hash digest d and the user's private state st_U .
- $\text{Auth}(\text{pk}, \Delta, d, \text{st}_A)$: The authentication algorithm is probabilistic, takes as input pk , the (global) secret key $\Delta \in \mathcal{R}$, hash digest d and secret state st_A , outputs a compressive encoding e of the MAC.
- $\text{Ext}(\text{pk}, e, \text{st}_U)$: The extension algorithm is deterministic, takes as input pk , the encoding e and secret state st_U , outputs the MAC σ for $\mathbf{x}$.

We additionally require our batch-MAC to satisfy several properties: succinctness, correctness, message privacy, and key privacy. Informally, succinctness states that the total communication of the protocol, or rather, the overall size of d and e , scales sublinearly with the message length $m = |\mathbf{x}|$. We formally define correctness, message privacy, and key privacy as follows.

Definition 4 (Correctness). A batch-MAC scheme $(\text{Setup}, \text{KeyGen}, \text{Hash}, \text{Auth}, \text{Ext})$ is correct if for any $\mathbf{x} \in \mathcal{R}^m$ and $\Delta \in \mathcal{R}$,

$$\Pr \left[\begin{array}{l} \text{pk} \xleftarrow{\$} \text{Setup}(1^\lambda) \\ (\mathbf{k}, \text{st}_A) \xleftarrow{\$} \text{KeyGen}(\text{pk}) \\ \sigma \neq \mathbf{x} \cdot \Delta + \mathbf{k} \\ \text{(over } \mathbb{Z}) : (d, \text{st}_U) \xleftarrow{\$} \text{Hash}(\text{pk}, \mathbf{x}) \\ e \xleftarrow{\$} \text{Auth}(\text{pk}, \Delta, d, \text{st}_A) \\ \sigma \leftarrow \text{Ext}(\text{pk}, e, \text{st}_U) \end{array} \right] \leq \text{negl}(\lambda).$$

Definition 5 (Statistical Message Privacy). A batch-MAC scheme $(\text{Setup}, \text{KeyGen}, \text{Hash}, \text{Auth}, \text{Ext})$ satisfies statistical message privacy, if for any $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{R}^m$, the following two distributions are statistically close:

$$\left\{ d \mid \begin{array}{l} \text{pk} \xleftarrow{\$} \text{Setup}(1^\lambda) \\ (d, \text{st}_U) \xleftarrow{\$} \text{Hash}(\text{pk}, \mathbf{x}_0) \end{array} \right\}, \left\{ d \mid \begin{array}{l} \text{pk} \xleftarrow{\$} \text{Setup}(1^\lambda) \\ (d, \text{st}_U) \xleftarrow{\$} \text{Hash}(\text{pk}, \mathbf{x}_1) \end{array} \right\}.$$

Definition 6 (Key Privacy). Consider the following experiment $\text{Exp}_{\text{batchMAC}}^{\mathcal{A}, \text{Sim}}(1^\lambda)$ parametrized by an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ and a simulator Sim :

1. Sample the public key $\text{pk} \xleftarrow{\$} \text{Setup}(1^\lambda)$.
2. Activate the adversary $(\mathbf{x}, \Delta, \text{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda, \text{pk})$.
3. Compute $(d, \text{st}_U) \xleftarrow{\$} \text{Hash}(\text{pk}, \mathbf{x})$ and use r_U to denote the randomness consumed.
4. Sample a random bit $b \xleftarrow{\$} \{0, 1\}$.
5. If $b = 0$ compute $(\mathbf{k}, \text{st}_A) \xleftarrow{\$} \text{KeyGen}(\text{pk})$ and $e_0 \xleftarrow{\$} \text{Auth}(\text{pk}, \Delta, d, \text{st}_A)$, else compute $e_1 \xleftarrow{\$} \text{Sim}(1^\lambda, \text{pk}, \mathbf{x}, r_U)$.
6. Compute $b' \leftarrow \mathcal{A}_1(e_b, r_U, \text{aux})$.
7. Return 1 if and only if $b' = b$.

We say that a batch-MAC scheme $(\text{Setup}, \text{KeyGen}, \text{Hash}, \text{Auth}, \text{Ext})$ satisfies key privacy, if there exists a PPT simulator Sim , such that for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\left| \Pr \left[\text{Exp}_{\text{batchMAC}}^{\mathcal{A}, \text{Sim}} = 1 \right] - \frac{1}{2} \right| \leq \text{negl}(\lambda).$$

3.1 DCR-based Construction of Batch-MAC

Now we show our first construction of batch-MAC, based on the DCR assumption. Our construction works for message space $\mathbb{Z}_B^m$ and key space $\mathbb{Z}_B$, where $B(\lambda)$ is the magnitude bound of $\mathbf{x}$ and Δ . We set $m = m_0 \cdot m_1$. To achieve our claimed complexity, take $m_0 = m^{2/3}$ and $m_1 = m^{1/3}$.

Construction 7 (Batch-MAC from DCR) *Construct*

$$\text{batchMAC} = (\text{Setup}, \text{KeyGen}, \text{Hash}, \text{Auth}, \text{Ext})$$

as follows:

– **Setup**(1^λ):

1. Sample public parameters for two Paillier groups.

$$\begin{aligned} \text{pp}_0 &:= (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^{\lambda_0}), \\ \text{pp}_1 &:= (G_1, F_1, H_1, f_1, N_1, \ell_1) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^{\lambda_1}), \end{aligned}$$

where $\lambda_0 > \lambda + \log(B^2)$ and $\lambda_1 > \lambda_0 + \log(m \cdot B \cdot \ell)$. Note that $N > 2^\lambda \cdot B^2$ and $N_1 > 2^\lambda \cdot mB\ell$.

2. Sample $m_1 + 1$ random group elements of G_1 . $(g_i, \rho_i) \xleftarrow{\$} \mathcal{D}(1^{\lambda_1}, \text{pp}_1)$ for $i = 0, 1, \dots, m_1$.

3. Output $\text{pk} := ((g_i, \rho_i)_{i=0,1,\dots,m_1}, \text{pp}_0, \text{pp}_1)$.

– **KeyGen**(pk):

1. Sample random vector $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_{N_1}^{m_0}$.

2. Sample m_1 random group elements of G . $(t_i, \tau_i) \xleftarrow{\$} \mathcal{D}(1^{\lambda_0}, \text{pp})$ for $i = 1, \dots, m_1$.

3. Compute

$$\mathbf{k}[(j-1) \cdot m_1 + i] := \text{DDLog}(\text{pp}, t_i^{-\mathbf{r}[j]})$$

for $i = 1, \dots, m_1$ and $j = 1, \dots, m_0$.

4. Output the key $\mathbf{k}$.

5. Output the private state as $\text{st}_A := ((t_i, \tau_i)_{i=1,\dots,m_1}, \mathbf{r})$.

– **Hash**($\text{pk}, \mathbf{x}$):

1. Split $\mathbf{x}$ into m_0 pieces, each of length m_1 . That is,

$$\begin{aligned} \mathbf{x} &= (\mathbf{x}_1, \dots, \mathbf{x}_{m_0}), \\ \mathbf{x}_j &= \mathbf{x}[(j-1) \cdot m_1 + 1, \dots, (j-1) \cdot m_1 + m_1], \text{ for } j = 1, \dots, m_0. \end{aligned}$$

2. Sample random values $u_j \xleftarrow{\$} [\ell_1]$ for $j = 1, \dots, m_0$.

3. Output d as a length- m_0 vector

$$\begin{aligned} d &:= (d_1, \dots, d_{m_0}), \\ d_j &:= g_0^{u_j} \cdot \prod_{i=1}^{m_1} g_i^{\mathbf{x}_j[i]}, \text{ for } j = 1, \dots, m_0. \end{aligned}$$

4. Output the private state as $\text{st}_U := (\mathbf{x}, (u_j)_{j=1,\dots,m_0})$.

– Auth(pk, Δ , d , st_A):

1. Sample random vector $\mathbf{w} \xleftarrow{\$} [\ell]^{m_1}$.

2. Sample random value $s \xleftarrow{\$} [\ell_1]$.

3. Compute

$$e_1 := (e_1^0, e_1^1, \dots, e_1^{m_1}),$$

$$e_1^0 := g_0^s,$$

$$e_1^i := g_i^s \cdot f_1^{\mathbf{w}[i]}, \text{ for } i = 1, \dots, m_1.$$

4. Parse d as $(d_1, \dots, d_{m_0})$. Parse $\mathbf{r}$ from st_A.

5. Compute

$$e_2 := (e_2^1, \dots, e_2^{m_0}),$$

$$e_2^j := \mathbf{r}[j] - \text{DDLog}(\text{pp}_1, d_j^s) \bmod N_1, \text{ for } j = 1, \dots, m_0.$$

6. Set $e_3 := (t_i, \tau_i)_{i=1, \dots, m_1}$.

7. Compute

$$e_4 := (e_4^{i,j})_{i=1, \dots, m_1; j=1, \dots, m_1},$$

$$e_4^{i,j} := t_j^{\mathbf{w}[i]} \cdot f^{\Delta \cdot \delta_{i,j}},$$

where $\delta_{i,j} = 1$ if $i = j$, else $\delta_{i,j} = 0$.

8. Output $e := (e_1, e_2, e_3, e_4)$.

– Ext(pk, e , st_U):

1. Compute a vector

$$\mathbf{v}' := (v'_1, \dots, v'_{m_0}),$$

$$v'_j := \text{DDLog}(\text{pp}_1, v''_j),$$

$$v''_j := (e_1^0)^{u_j} \cdot \prod_{i=1}^{m_1} (e_1^i)^{\mathbf{x}_j[i]}, \text{ for } j = 1, \dots, m_0.$$

2. Compute $\mathbf{v} := \mathbf{v}' + e_2$ over $\mathbb{Z}_{N_1}^{m_0}$.

3. Compute

$$\sigma_0[(j-1) \cdot m_1 + i] := t_i^{-\mathbf{v}[j]}$$

for $i = 1, \dots, m_1$ and $j = 1, \dots, m_0$.

4. Compute

$$\sigma_1[(j-1) \cdot m_1 + i] := \prod_{\kappa=1}^{m_1} (e_4^{\kappa,i})^{\mathbf{x}_j[\kappa]}$$

for $i = 1, \dots, m_1$ and $j = 1, \dots, m_0$.

5. Compute

$$\sigma[(j-1) \cdot m_1 + i] := \text{DDLog}\left(\text{pp}, \sigma_0[(j-1) \cdot m_1 + i] \cdot \sigma_1[(j-1) \cdot m_1 + i]\right)$$

for $i = 1, \dots, m_1$ and $j = 1, \dots, m_0$.

6. Output the MAC σ .

Communication Cost of Construction 7. We take $m_0 = m^{2/3}$ and $m_1 = m^{1/3}$. The public key $\mathbf{pk}$ is of size $O(m^{2/3}\lambda)$. The hash digest d consists of m_0 group elements, of total size $O(m^{2/3}\lambda)$. The MAC encoding e consists of $O(m_0 + m_1^2)$ group elements, of total size $O(m^{2/3}\lambda)$. Thus, Construction 7 achieves succinctness. Moreover, the user's randomness within **Hash** is $(u_j)_{j=1,\dots,m_0}$, which is of size $O(m^{2/3}\lambda)$. The authenticator's randomness within **KeyGen** is $\mathbf{st}_A$, which is of size $O(m^{2/3}\lambda)$.

The following theorem formally establishes the correctness, message privacy, and key privacy of Construction 7 as a batch-MAC.

Theorem 8 (Batch-MAC from DCR). *Assuming the DCR assumption, the batch-MAC scheme in Construction 7 satisfies correctness, statistical message privacy, and key privacy.*

Proof of Correctness. We first prove that

$$\mathbf{v} = \mathbf{r} + (\langle \mathbf{x}_1, \mathbf{w} \rangle, \dots, \langle \mathbf{x}_{m_0}, \mathbf{w} \rangle) \text{ over } \mathbb{Z}_{N_1}^{m_0}.$$

First, note that for any j in $\{1, \dots, m_0\}$

$$\begin{aligned} v_j'' / d_j^s &= (e_1^0)^{u_j} \cdot \prod_{i=1}^{m_1} (e_1^i)^{\mathbf{x}_j[i]} / (g_0^{u_j} \cdot \prod_{i=1}^{m_1} g_i^{\mathbf{x}_j[i]})^s \\ &= (g_0^s)^{u_j} \cdot \prod_{i=1}^{m_1} (g_i^s \cdot f_1^{\mathbf{w}[i]})^{\mathbf{x}_j[i]} / (g_0^{u_j} \cdot \prod_{i=1}^{m_1} g_i^{\mathbf{x}_j[i]})^s \\ &= \prod_{i=1}^{m_1} (f_1^{\mathbf{w}[i]})^{\mathbf{x}_j[i]} \\ &= f_1^{\langle \mathbf{w}, \mathbf{x}_j \rangle}. \end{aligned}$$

Thus, for any j in $\{1, \dots, m_0\}$

$$\begin{aligned} v_j' - \text{DDLog}(\mathbf{pp}_1, d_j^s) &= \text{DDLog}(\mathbf{pp}_1, v_j'') - \text{DDLog}(\mathbf{pp}_1, d_j^s) \\ &= \langle \mathbf{w}, \mathbf{x}_j \rangle \text{ (over } \mathbb{Z}_{N_1}). \end{aligned}$$

Combining with $e_2^j := \mathbf{r}[j] - \text{DDLog}(\mathbf{pp}_1, d_j^s) \bmod N_1$, we get

$$\begin{aligned} \mathbf{v}[j] &= v_j' + e_2^j \\ &= (\langle \mathbf{w}, \mathbf{x}_j \rangle + \text{DDLog}(\mathbf{pp}_1, d_j^s)) + (\mathbf{r}[j] - \text{DDLog}(\mathbf{pp}_1, d_j^s)) \\ &= \mathbf{r}[j] + \langle \mathbf{w}, \mathbf{x}_j \rangle \end{aligned}$$

over $\mathbb{Z}_{N_1}$, for any j in $\{1, \dots, m_0\}$. This gives $\mathbf{v} = \mathbf{r} + (\langle \mathbf{x}_1, \mathbf{w} \rangle, \dots, \langle \mathbf{x}_{m_0}, \mathbf{w} \rangle)$ over $\mathbb{Z}_{N_1}^{m_0}$. Next, note that

$$|\langle \mathbf{x}_{m_0}, \mathbf{w} \rangle| < m \cdot B \cdot \ell < \frac{1}{2^\lambda} \cdot N_1.$$

Thus $\mathbf{v} = \mathbf{r} + (\langle \mathbf{x}_1, \mathbf{w} \rangle, \dots, \langle \mathbf{x}_{m_0}, \mathbf{w} \rangle)$ holds in $\mathbb{Z}^{m_0}$ with overwhelming probability. Consequently, for any $i \in \{1, \dots, m_1\}$ and $j \in \{1, \dots, m_0\}$

$$\begin{aligned}
& \sigma_0[(j-1) \cdot m_1 + i] \cdot \sigma_1[(j-1) \cdot m_1 + i] / t_i^{-\mathbf{r}[j]} \\
&= t_i^{-\mathbf{v}[j]} \cdot \prod_{\kappa=1}^{m_1} (e_4^{\kappa, i})^{\mathbf{x}_j[\kappa]} / t_i^{-\mathbf{r}[j]} \\
&= t_i^{-\mathbf{v}[j]} \cdot \prod_{\kappa=1}^{m_1} (t_i^{\mathbf{w}[\kappa]} \cdot f^{\Delta \cdot \delta_{\kappa, i}})^{\mathbf{x}_j[\kappa]} / t_i^{-\mathbf{r}[j]} \\
&= t_i^{-\mathbf{v}[j]} / t_i^{-\mathbf{r}[j]} \cdot \prod_{\kappa=1}^{m_1} t_i^{\mathbf{w}[\kappa] \cdot \mathbf{x}_j[\kappa]} \cdot f^{\Delta \cdot \mathbf{x}_j[i]} \\
&= t_i^{-\langle \mathbf{x}_j, \mathbf{w} \rangle} \cdot t_i^{\langle \mathbf{w}, \mathbf{x}_j \rangle} \cdot f^{\Delta \cdot \mathbf{x}_j[i]} \\
&= f^{\Delta \cdot \mathbf{x}_j[i]}.
\end{aligned}$$

Thus, for any $i \in \{1, \dots, m_1\}$ and $j \in \{1, \dots, m_0\}$

$$\begin{aligned}
& \sigma[(j-1) \cdot m_1 + i] - \mathbf{k}[(j-1) \cdot m_1 + i] \\
&= \text{DDLog}(\text{pp}, \sigma_0[(j-1) \cdot m_1 + i] \cdot \sigma_1[(j-1) \cdot m_1 + i]) - \text{DDLog}(\text{pp}, t_i^{-\mathbf{r}[j]}) \\
&= \Delta \cdot \mathbf{x}_j[i] \text{ (over } \mathbb{Z}_N).
\end{aligned}$$

Given that $N > 2^\lambda \cdot |\Delta \mathbf{x}_j[i]|$, $\sigma = \mathbf{k} + \Delta \cdot \mathbf{x}$ holds in $\mathbb{Z}^m$ with overwhelming probability.

Proof of Statistical Message Privacy. Notice that within d , each hash value d_j is masked by a fresh random value $u_j \xleftarrow{\$} [\ell_1]$. Given that $g_0^{u_j}$ is statistically indistinguishable from a random element of G_1 , d is statistically indistinguishable from a uniform random vector in G^{m_0} .

Proof of Key Privacy. We prove by the hybrid argument. Consider the following hybrid worlds.

- Hybrid₀: This is the real world, which correspond to the $b = 0$ case in $\text{Exp}_{\text{batchMAC}}^{\mathcal{A}, \text{Sim}}(1^\lambda)$. The adversary's view for e consists of

$$\begin{aligned}
e_1 &= (g_0^s, g_1^s \cdot f_1^{\mathbf{w}[1]}, \dots, g_{m_1}^s \cdot f_1^{\mathbf{w}[m_1]}); \\
e_2 &= \mathbf{r} - (\text{DDLog}(\text{pp}_1, d_1^s), \dots, \text{DDLog}(\text{pp}_1, d_{m_0}^s)); \\
e_3 &= ((t_1, \tau_1), \dots, (t_{m_1}, \tau_{m_1})); \\
e_4 &= (t_j^{\mathbf{w}[i]} \cdot f^{\Delta \cdot \delta_{i,j}})_{i=1, \dots, m_1; j=1, \dots, m_1}.
\end{aligned}$$

- Hybrid₁: In this hybrid, we change e_2 to $\hat{\mathbf{r}} \xleftarrow{\$} \mathbb{Z}_{N_1}^{m_0}$. Hybrid₁ is identical to Hybrid₀ because $\mathbf{r}$ is uniformly random in $\mathbb{Z}_{N_1}^{m_0}$ and e_2 is computed over $\mathbb{Z}_{N_1}^{m_0}$. Note that in this hybrid, s only appears in e_1 .

- Hybrid₂: In this hybrid, we change e_1 to

$$e_1 = (g_0^s \cdot f_1^{\nu_0}, g_1^s \cdot f_1^{\nu_1} \cdot f_1^{\mathbf{w}[1]}, \dots, g_{m_1}^s \cdot f_1^{\nu_{m_1}} \cdot f_1^{\mathbf{w}[m_1]}),$$

where $\nu_0, \dots, \nu_{m_1} \xleftarrow{\$} \mathbb{Z}_{N_1}$.

Hybrid₂ is indistinguishable from Hybrid₁ by the m_1 -ary enhanced DDH assumption (Assumption 2) in the Paillier group G_1 .

- Hybrid₃: In this hybrid, e_1 is further changed to

$$e_1 = (g_0^s \cdot f_1^{\nu_0}, g_1^s \cdot f_1^{\nu_1}, \dots, g_{m_1}^s \cdot f_1^{\nu_{m_1}}),$$

where $\nu_0, \dots, \nu_{m_1} \xleftarrow{\$} \mathbb{Z}_{N_1}$.

Hybrid₃ is identical to Hybrid₂ by the randomness of $\nu_0, \dots, \nu_{m_1}$. Note that in this hybrid, $\mathbf{w}$ only appears in e_4 .

- Hybrid₄: In this hybrid, we change e_3 and e_4 to

$$e_3 = ((t_1, \tau_1), \dots, (t_{m_1}, \tau_{m_1})),$$

$$e_4 = (t_j^{\mathbf{w}[i]} \cdot f^{\mu_{i,j}} \cdot f^{\Delta \cdot \delta_{i,j}})_{i=1, \dots, m_1; j=1, \dots, m_1},$$

where $\mu_{i,j} \xleftarrow{\$} \mathbb{Z}_N$. Hybrid₄ is computationally indistinguishable from Hybrid₃ by the following lemma. It generalizes the enhanced DDH assumption to a matrix version. We defer the lemma's proof to Appendix A.

Lemma 4. *Assuming the DCR assumption, the following distributions are computationally indistinguishable.*

$$\left\{ \begin{array}{l} \text{pp} \\ t_1, \dots, t_n \\ \tau_1, \dots, \tau_n \\ t_1^{\mathbf{w}}, \dots, t_n^{\mathbf{w}} \end{array} \middle| \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [n] : (t_j, \tau_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ \mathbf{w} \xleftarrow{\$} [\ell]^n \end{array} \right\},$$

$$\left\{ \begin{array}{l} \text{pp} \\ t_1, \dots, t_n \\ \tau_1, \dots, \tau_n \\ f^{\mu_1} \odot t_1^{\mathbf{w}}, \dots, f^{\mu_n} \odot t_n^{\mathbf{w}} \end{array} \middle| \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [n] : (t_j, \tau_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ \mathbf{w} \xleftarrow{\$} [\ell]^n \\ \forall j \in [n] : \mu_j \xleftarrow{\$} \mathbb{Z}_N^n \end{array} \right\}.$$

- Hybrid₅: In this hybrid, we further change e_4 to

$$e_4 = (t_j^{\mathbf{w}[i]} \cdot f^{\mu_{i,j}})_{i=1, \dots, m_1; j=1, \dots, m_1}$$

where $\mu_{i,j} \xleftarrow{\$} \mathbb{Z}_N$.

Hybrid₅ is identical to Hybrid₄ by the randomness of $\mu_{i,j}$. Note that now in this hybrid, e is totally independent of Δ , and can be easily simulated.

3.2 LWE-based Construction of Batch-MAC

In this section, we construct a Batch-MAC scheme based on the LWE assumption. The communication cost of this scheme is only poly-logarithmic in the

message length. Our construction is obtained via an interesting observation on recent progress in succinct secret sharing.

As preparation, we recap the concept of non-interactive oblivious tensor evaluation (NI-OTE) in [2]. It enables two parties, holding vectors $\mathbf{x}$ and $\mathbf{y}$ respectively, to establish additive shares of $\mathbf{x} \otimes \mathbf{y}$ through a single round of simultaneous messages.

Definition 7 (NI-OTE of [2]). Let $m := m(\lambda)$, $\ell := \ell(\lambda)$, and $q := q(\lambda)$ be positive integers. An NI-OTE for $\mathbb{Z}_q^m \otimes \mathbb{Z}_q^\ell$ consists of a tuple of PPT algorithms (Setup, Hash, Enc, HashEval, EncEval) with the following syntax:

- Setup(1^λ): The setup algorithm is probabilistic, takes as input the security parameter 1^λ and outputs a public key $\mathbf{pk}$.
- Hash($\mathbf{pk}, \mathbf{x}$): The hashing algorithm is probabilistic and takes as input a public key $\mathbf{pk}$ and the description of a vector $\mathbf{x} \in \mathbb{Z}_q^m$. The output is a digest d and the hasher's private information ψ .
- Enc($\mathbf{pk}, \mathbf{y}$): The encoding algorithm is probabilistic and takes as input a public key $\mathbf{pk}$ and a vector $\mathbf{y} \in \mathbb{Z}_q^\ell$. The output is an encoding E and the encoder's private information ϕ .
- HashEval($\mathbf{pk}, E, \psi$): The hasher's evaluation algorithm is deterministic and takes as input a public key $\mathbf{pk}$, an encoding E , and the hasher's private information ψ . The output is a vector $\mathbf{s}_0 \in \mathbb{Z}_q^{m \cdot \ell}$.
- EncEval($\mathbf{pk}, d, \phi$): The encoder's evaluation algorithm is deterministic and takes as input a public key $\mathbf{pk}$, a digest d , and the encoder's private information ϕ . The output is a vector $\mathbf{s}_1 \in \mathbb{Z}_q^{m \cdot \ell}$.

Informally, for an NI-OTE scheme, correctness means that the hasher and encoder establish a pair of additive shares of $\mathbf{x} \otimes \mathbf{y}$. Hasher privacy means the encoder learns nothing about $\mathbf{x}$, and encoder privacy means the hasher learns nothing about $\mathbf{y}$.

Definition 8 (Correctness of NI-OTE). An NI-OTE scheme (Setup, Hash, Enc, HashEval, EncEval) is α -correct if there exists a negligible function $\text{negl}(\lambda)$ such that, for every $\mathbf{x} \in \mathbb{Z}_q^m$ and $\mathbf{y} \in \mathbb{Z}_q^\ell$, we have that:

$$\Pr \left[\|\mathbf{s}_0 + \mathbf{s}_1 - \mathbf{x} \otimes \mathbf{y}\|_\infty > \alpha \cdot \|\mathbf{x}\|_\infty \right] \leq \text{negl}(\lambda)$$

where the probability is taken over the random choice of $\mathbf{pk} \xleftarrow{\$} \text{Setup}(1^\lambda)$, $(d, \psi) \xleftarrow{\$} \text{Hash}(\mathbf{pk}, \mathbf{x})$, and $(E, \phi) \xleftarrow{\$} \text{Enc}(\mathbf{pk}, \mathbf{y})$.

Definition 9 (Statistical Hasher Privacy of NI-OTE). An NI-OTE scheme (Setup, Hash, Enc, HashEval, EncEval) satisfies statistical hasher privacy, if for any $\mathbf{x}_0$ and $\mathbf{x}_1$ in $\mathbb{Z}_q^m$, the following two distributions are statistically close:

$$\left\{ d \left| \begin{array}{l} \mathbf{pk} \xleftarrow{\$} \text{Setup}(1^\lambda) \\ (d, \psi) \xleftarrow{\$} \text{Hash}(\mathbf{pk}, \mathbf{x}_0) \end{array} \right. \right\}, \left\{ d \left| \begin{array}{l} \mathbf{pk} \xleftarrow{\$} \text{Setup}(1^\lambda) \\ (d, \psi) \xleftarrow{\$} \text{Hash}(\mathbf{pk}, \mathbf{x}_1) \end{array} \right. \right\}.$$

Definition 10 (Encoder Privacy of NI-OTE). Consider the following experiment $\text{Exp}_{\text{OTE}}^{\mathcal{A}, \text{Sim}}(1^\lambda)$ parametrized by an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ and a simulator Sim :

1. Sample a public key $\text{pk} \xleftarrow{\$} \text{Setup}(1^\lambda)$.
2. Activate the adversary $(\mathbf{y}, \text{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda, \text{pk})$.
3. Sample a random bit $b \xleftarrow{\$} \{0, 1\}$.
4. If $b = 0$, compute $(E_0, \phi_0) \xleftarrow{\$} \text{Enc}(\text{pk}, \mathbf{y})$, else compute $E_1 \xleftarrow{\$} \text{Sim}(1^\lambda, \text{pk})$.
5. Compute $b' \leftarrow \mathcal{A}_1(E_b, \text{aux})$.
6. Return 1 if and only if $b = b'$.

We say that an NI-OTE scheme $(\text{Setup}, \text{Hash}, \text{Enc}, \text{HashEval}, \text{EncEval})$ is encoder-private if there exists a PPT simulator Sim , such that for every PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that:

$$\left| \frac{1}{2} - \Pr \left[\text{Exp}_{\text{OTE}}^{\mathcal{A}, \text{Sim}}(1^\lambda) = 1 \right] \right| \leq \text{negl}(\lambda).$$

Remark 1. The original definition of NI-OTE in [2] does not include hasher privacy, which is necessary for message privacy of our batch-MAC. Fortunately, given that the hashing algorithm is a compression, achieving statistical privacy is easy. We show a slight modification of [2]’s NI-OTE with statistical hasher privacy in Appendix B.

Additionally, the NI-OTE construction in [2] satisfies a critical property they call *bilinear encoder evaluation*. It requires that

1. The digest d is a vector in $\mathbb{Z}_q^n$.
2. The encoder secret information ϕ is a vector in $\mathbb{Z}_q^k$.
3. The encoder evaluation algorithm consists of

$$\text{EncEval}(\text{pk}, d, \phi) := \mathbf{P} \cdot (d \otimes \phi \otimes \mathbf{g}_q^\top)$$

where the matrix $\mathbf{P} \in \mathbb{Z}_q^{(m \cdot \ell) \times (n \cdot k \cdot \log q)}$ can be publicly derived from pk .

We summarize the NI-OTE scheme of [2] as the following lemma.

Lemma 5 (NI-OTE of [2] from LWE). Assuming the LWE assumption, for integers $m = t^r \cdot n$, $n = k \log q$, $k = \Theta(\lambda)$, there exists an NI-OTE scheme for $\mathbb{Z}_q^m \otimes \mathbb{Z}_q^n$ with α -correctness, statistical hasher privacy, encoder privacy, and bilinear encoder evaluation.

Furthermore, it has the following properties:

- The encoder’s public decoding matrix $\mathbf{P}$ satisfies $\|\mathbf{P}\|_\infty \leq n^{2r}$.
- The digest d satisfies $\|d\|_\infty \leq (tn)^r \|\mathbf{x}\|_\infty$.
- The correctness error $\alpha \leq tn(n+1)E(\lambda)(tn^3)^r$, where $E(\lambda)$ is the upper bound for the infinity norm of errors in LWE samples.

Moreover, the scheme is fully succinct:

- the digest d is of size $n \cdot (r \cdot \log tn + \log \|\mathbf{x}\|_\infty) = \text{poly}(\log m, \lambda)$;
- the encoding E is of size $r \cdot (tn)^2 \cdot n \log q = \text{poly}(\log m, \lambda)$.

Now we construct our batch-MAC scheme. Our batch-MAC works for the bounded integer setting; that is, for message space $\mathbb{Z}_B^m$ and key space $\mathbb{Z}_B$. $B(\lambda)$ is a magnitude bound satisfying $B^2(\lambda) \ll p(\lambda) \ll q(\lambda)$. To be more precise, $p|q$, $p \geq B^2 \cdot 2^\lambda$, $q \geq p \cdot \lambda^{\omega(\lambda)}$.

Let m have the form of $m = t^r \cdot n$ and $n = k \log q$, $k = \Theta(\lambda)$ (else padding 0s to the end). We assume the existence of an NI-OTE scheme $\text{OTE} = (\text{Setup}, \text{Hash}, \text{Enc}, \text{HashEval}, \text{EncEval})$ for $\mathbb{Z}_q^m \otimes \mathbb{Z}_q^n$ as described in Lemma 5.

Construction 9 (LWE-based Batch-MAC) Construct $\text{batchMAC} = (\text{Setup}, \text{KeyGen}, \text{Hash}, \text{Auth}, \text{Ext})$ as follows:

- $\text{Setup}(1^\lambda)$:
 1. Compute $\text{pk}_{\text{OTE}} \xleftarrow{\$} \text{OTE.Setup}(1^\lambda)$.
 2. Sample a random matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{k \times n^2}$.
 3. Output $\text{pk} = (\text{pk}_{\text{OTE}}, \mathbf{A})$.
- $\text{KeyGen}(\text{pk})$:
 1. Sample a random vector $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$.
 2. Compute the public decoding matrix of OTE out of pk_{OTE} and denote it as $\mathbf{P} \in \mathbb{Z}_q^{mn \times n^2}$.
 3. Compute $\mathbf{k}_0 := \mathbf{P} \cdot \mathbf{A}^\top \cdot \mathbf{s}$.
 4. Concatenate the entries of $\mathbf{k}_0$ of index $1, n+1, 2n+1, \dots, (m-1)n+1$, as a vector $\mathbf{k}_1 \in \mathbb{Z}_q^m$.
 5. Output the MAC key $\mathbf{k} := \lfloor \mathbf{k}_1 \rfloor_{q/p}$.
 6. Output the private state $\text{st}_A := \mathbf{s}$.
- $\text{Hash}(\text{pk}, \mathbf{x})$:
 1. Compute $(\mathbf{d}, \psi) \xleftarrow{\$} \text{OTE.Hash}(\text{pk}_{\text{OTE}}, \mathbf{x})$.
 2. Output the hash digest $d := \mathbf{d}$.
 3. Output the private state $\text{st}_U := \psi$.
- $\text{Auth}(\text{pk}, \Delta, d, \text{st}_A)$:
 1. Sample an LWE error $\mathbf{e} \xleftarrow{\$} \chi(\lambda)$.
 2. Set $\Delta' = \Delta \cdot \frac{q}{p}$.
 3. Compute

$$(E, \mathbf{r}) \xleftarrow{\$} \text{OTE.Enc}(\text{pk}_{\text{OTE}}, (\Delta', \mathbf{0})),$$
 where $(\Delta', \mathbf{0})$ means a length- n vector acquired by appending 0s to Δ' .
 4. Set

$$\begin{aligned} e_0 &:= \mathbf{A}^\top \mathbf{s} + \mathbf{e} + \mathbf{d} \otimes \mathbf{r} \otimes \mathbf{g}_q^\top, \\ e_1 &:= E. \end{aligned}$$
 5. Output $e := (e_0, e_1)$.
- $\text{Ext}(\text{pk}, e, \text{st}_U)$:
 1. Compute the public decoding matrix $\mathbf{P}$ of OTE out of pk_{OTE} .
 2. Compute

$$\begin{aligned} \sigma_0 &\leftarrow \text{OTE.HashEval}(\text{pk}_{\text{OTE}}, e_1, \text{st}_U), \\ \sigma_1 &:= \mathbf{P} \cdot e_0 + \sigma_0. \end{aligned}$$
 3. Concatenate the entries in σ_1 of index $1, n+1, 2n+1, \dots, (m-1)n+1$ as a vector $\sigma_2 \in \mathbb{Z}_q^m$.
 4. Output the MAC $\sigma := \lfloor \sigma_2 \rfloor_{q/p}$.

Communication Cost of Construction 9: Given that the underlying OTE scheme has digest size $\text{poly}(\log m, \lambda)$ and encoding size $\text{poly}(\log m, \lambda)$, as shown in Lemma 5. Then, in Construction 9, the hash digest size $|d| \leq \text{poly}(\log m, \lambda)$ and the encoding size

$$|e| \leq n^2 \log q + \text{poly}(\log m, \lambda) \leq \text{poly}(\log m, \lambda).$$

Moreover, let $\mathbf{r}_A$ and $\mathbf{r}_U$ denote the randomness consumed within algorithm KeyGen and Hash respectively. We have $|\mathbf{r}_A| = |\mathbf{s}| \cdot \log q \leq \text{poly}(\lambda)$ and $|\mathbf{r}_U| \leq \text{poly}(\lambda)$ ². In conclusion, the total communication is $\text{poly}(\log m, \lambda)$, and the randomness size of KeyGen and Hash is $\text{poly}(\lambda)$.

The following theorem formally establishes the correctness, message privacy, and key privacy of Construction 9 as a batch-MAC.

Theorem 10. *Assuming the LWE assumption, the batchMAC scheme in Construction 9 satisfies correctness, statistical sender privacy, and receiver privacy.*

Proof of Correctness. By the bilinear encoder evaluation property of OTE,

$$\begin{aligned} \sigma_1 &= \mathbf{P} \cdot e_0 + \sigma_0 \\ &= \mathbf{P} \cdot \mathbf{A}^\top \mathbf{s} + \mathbf{P} \cdot \mathbf{e} + \mathbf{P} \cdot (\mathbf{d} \otimes \mathbf{r} \otimes \mathbf{g}_q^\top) + \sigma_0 \\ &= \mathbf{P} \cdot \mathbf{A}^\top \mathbf{s} + \mathbf{P} \cdot \mathbf{e} + \mathbf{w} + \sigma_0, \end{aligned}$$

where $\mathbf{w} \leftarrow \text{OTE.EncEval}(\text{pk}_{\text{OTE}}, \mathbf{d}, \mathbf{r})$. By the α -correctness of OTE,

$$\mathbf{w} + \sigma_0 = \mathbf{x} \otimes (\Delta', \mathbf{0}) + \mathbf{e}',$$

where $\|\mathbf{e}'\|_\infty \leq \alpha \cdot \|\mathbf{x}\|_\infty$. Thus

$$\begin{aligned} \sigma_1 &= \mathbf{P} \cdot \mathbf{A}^\top \mathbf{s} + \mathbf{x} \otimes (\Delta', \mathbf{0}) + (\mathbf{P} \cdot \mathbf{e} + \mathbf{e}') \\ &= \mathbf{k}_0 + \mathbf{x} \otimes (\Delta', \mathbf{0}) + \mathbf{e}_0, \end{aligned}$$

where

$$\begin{aligned} \|\mathbf{e}_0\|_\infty &= \|\mathbf{P} \cdot \mathbf{e} + \mathbf{e}'\|_\infty \\ &\leq n^2 \cdot E(\lambda) \cdot \|\mathbf{P}\|_\infty + \alpha \cdot \|\mathbf{x}\|_\infty \\ &\leq n^2 \cdot E(\lambda) \cdot n^{2r} + tn(n+1)E(\lambda)(tn^3)^r \cdot B(\lambda) \\ &\ll q/p, \end{aligned}$$

where $p/q \in \lambda^{-\omega(\lambda)}$. By taking the entries of index $1, n+1, 2n+1, \dots, (m-1)n+1$, we have

$$\sigma_2 - \mathbf{k}_1 = \mathbf{x} \cdot \left(\Delta \cdot \frac{q}{p} \right) + \mathbf{e}_1,$$

where $\|\mathbf{e}_1\|_\infty = \|\mathbf{e}_0\|_\infty \ll q/p$. Thereby

$$\sigma - \mathbf{k} = \lfloor \sigma_2 \rfloor_{q/p} - \lfloor \mathbf{k}_1 \rfloor_{q/p} = \mathbf{x} \cdot \Delta$$

holds in $\mathbb{Z}_p^m$ with overwhelming probability. Given that $p \gg B^2 \geq \|\mathbf{x}\Delta\|_\infty$, $\sigma - \mathbf{k} = \mathbf{x} \cdot \Delta$ holds in $\mathbb{Z}^m$ with overwhelming probability.

² $\mathbf{r}_U$ is exactly the randomness used in OTE.Hash . It will be shown to be of size $\text{poly}(\lambda)$ in our modified construction of NI-OTE, in Appendix B.

Proof of Statistical Message Privacy. The statistical message privacy of `batchMAC` follows directly from the hasher privacy of OTE.

Proof of Key Privacy. Let the simulator `Sim` do as follows:

- Sample a uniformly random vector $\mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^{n^2}$ as e_0 ;
- Let `Sim'` be the simulator in the encoder privacy experiment $\text{Exp}_{\text{OTE}}^{\mathcal{A}, \text{Sim}}(1^\lambda)$ of the underlying OTE scheme. Output $E \xleftarrow{\$} \text{Sim}'(1^\lambda, \text{pk}_{\text{OTE}})$ as e_1 .

We argue that this experiment is indistinguishable from the original experiment through the following sequence of hybrids:

- `Hybrid0`: This is the original experiment, where $e_0 = \mathbf{A}^\top \mathbf{s} + \mathbf{e} + \mathbf{d} \otimes \mathbf{r} \otimes \mathbf{g}_q^\top$ with $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, and $e_1 = \text{OTE}.\text{Enc}(\text{pk}_{\text{OTE}}, (\Delta', \mathbf{0}))$.
- `Hybrid1`: Provide $\mathcal{A}$ with $\mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^{n^2}$ as e_0 , while e_1 remains the same. This is indistinguishable from `Hybrid0` by a direct reduction to the LWE assumption, where in particular $\mathbf{A}^\top \mathbf{s} + \mathbf{e}$ is an LWE sample.
- `Hybrid2`: Further substitute e_1 as $E \xleftarrow{\$} \text{Sim}'(1^\lambda, \text{pk}_{\text{OTE}})$. This is indistinguishable from `Hybrid1` by the encoder privacy of OTE.

Thus we finish the proof.

4 Circuit Argument of Knowledge, a Heuristic Roadmap

In a typical VOLE-based ZK proof for circuit C , a VOLE for witness $\mathbf{w}$ is needed between the prover and the verifier, where $|\mathbf{w}| \approx |C|$. Thus, it requires at least $|C|$ communication to establish this VOLE, which turns out to be the bottleneck. We observe that we can plug in our batch-MAC scheme to generate the required VOLE. The consequent proof has only polylogarithmic size and uses only 3 rounds. This provides a new roadmap for constructing lattice-based succinct AoKs. Moreover, our AoK enjoys two merits. First, the verifier storage scales only with the memory required to verify the statement in the clear. Second, the verifier can transfer the heavy circuit-sized computation to an offline phase before the protocol. In this case, the online verifier time is almost independent of the circuit size. In this section, we show a relatively complete description of our circuit AoK.

Our AoK is based on QuickSilver [69], which is a VOLE-based zero-knowledge proof for arithmetic circuits. Suppose the prover P has a private witness $\mathbf{w} \in \mathbb{F}_Q^n$, and it wants to convince the verifier V that $C(\mathbf{w}) = 1$ for a public statement circuit C . Suppose C has t multiplication gates. P sets the trace of the circuit evaluation as $\mathbf{x} = (\mathbf{w}, y_1, \dots, y_t, \rho)$, where y_i is the output of the i -th multiplication gate and $\rho \xleftarrow{\$} \mathbb{F}_Q$ is an independent mask value. The QuickSilver protocol goes as follows:

1. Establish IT-MACs for the trace $\mathbf{x}$, where V acts the authenticator. More formally, suppose V has MAC keys $\Delta \in \mathbb{F}_Q$ and $\mathbf{k} \in \mathbb{F}_Q^{(n+t+1)}$, then, P has

MAC tags $\sigma = \mathbf{k} + \Delta \cdot \mathbf{x}$. Conventionally, these MACs are established by invoking a VOLE-PCG and then sending a derandomization message from P to V . This causes two drawbacks: the proof size must be at least linear, and V only acquires $\mathbf{k}$ after the derandomization.

2. For each affine gate g_{aff} , if the MACs for its input wires are already established, P and V can locally derive a MAC for its output wire by leveraging the affine homomorphism of IT-MACs. For each multiplication gate g_{mul} , V has to check the consistency between its inputs and outputs. Suppose

$$\begin{aligned} \text{Gate } g_{\text{mul}} : a \times b &\mapsto c; \\ V : \Delta, k_a, k_b, k_c; \\ P : \sigma_a &= a \cdot \Delta + k_a, \sigma_b = b \cdot \Delta + k_b, \sigma_c = c \cdot \Delta + k_c. \end{aligned}$$

To check the consistency between a , b , and c :

$$\begin{aligned} V : k_a \cdot k_b + \Delta \cdot k_c \\ &= (\sigma_a - a\Delta)(\sigma_b - b\Delta) + \Delta(\sigma_c - c\Delta) \\ &= (\sigma_a \cdot \sigma_b) - \Delta \cdot (a\sigma_b + b\sigma_a - \sigma_c) + \Delta^2 \cdot (a \cdot b - c) \\ P : (\sigma_a \cdot \sigma_b), \quad (a\sigma_b + b\sigma_a - \sigma_c). \end{aligned}$$

3. Note that if g_{mul} is honestly calculated, $(a \cdot b - c)$ should be 0. In this case, P and V should satisfy a “linear relation” on Δ , not a quadratic one. In other words, V has to verify t linear relations. These can be checked by a batched opening:

$$\begin{aligned} V : \text{Sample } \chi &\xleftarrow{\$} \mathbb{F}_Q; \quad V \rightarrow P : \chi; \\ V : \text{Compute } F &:= \sum_{i \in [t]}^{\text{for } t \text{ mult gates}} (k_a^i \cdot k_b^i + \Delta \cdot k_c^i) \cdot \chi^i; \\ P : \text{Compute } A &:= \sum_{i \in [t]} (\sigma_a^i \cdot \sigma_b^i) \cdot \chi^i, \quad B := \sum_{i \in [t]} (a^i \sigma_b^i + b^i \sigma_a^i - \sigma_c^i) \cdot \chi^i. \\ P \rightarrow V : A + \sigma_\rho, \quad B + \rho, \quad \sigma_h. \end{aligned}$$

Where h is the output wire. Then, V checks:

$$(A + \sigma_\rho) - \Delta(B + \rho) = F + k_\rho, \quad \sigma_h = \Delta + k_h.$$

We observe that the MACs $\sigma = \mathbf{k} + \Delta \mathbf{x}$ in Step 1 can be established directly by our batch-MAC scheme, without linear derandomization. Specifically, invoking our LWE-based batch-MAC only costs $\text{poly}(\log(n + t + 1), \lambda)$ communication in Step1. In this case, the total communication of this circuit argument is only $\text{poly}(\log(n + t + 1), \lambda)$, and it has only 3 rounds. Moreover, V can set $\mathbf{k}$, Δ , χ before the protocol begins. Thus, all the heavy computations of V (the gate-by-gate computations in Step 2&3) can be done offline, making the online verification time only polylogarithmic in the circuit size (batchMAC.Auth costs only $\text{poly}(\log(n + t + 1), \lambda)$ time).

Furthermore, our circuit argument maintains a significant merit of VOLE-based ZK — the verifier only needs $O(M)$ memory, where M is the memory required to verify the statement in the clear. To see this, recap that to generate $\mathbf{k}$ in our LWE-based batch-MAC (algorithm **KeyGen** in Construction 9), V computes a linear function on a short seed $\mathbf{s} \mapsto \mathbf{G}\mathbf{s}$, where $\mathbf{G}$ is $(n+t+1) \times \ell$ and $\ell \in \text{poly}(\lambda)$. Let $m = (n+t+1)$, and suppose $m = 2^r \cdot \ell$. Then $\mathbf{G}$ has the form of $\mathbf{G} = \prod_{i=1}^r \mathbf{P}_i$, where $\mathbf{P}_i = (\mathbf{I}_{2^{r-i}} \otimes \mathbf{P})$ and $\mathbf{P}$ is $(2 \cdot \ell) \times \ell$. The j -th row of G can be computed within memory $O(r \cdot \ell)$ instead of $O(2^r \cdot \ell)$. Thus, the memory required to compute $\mathbf{k}[j]$ for each $j \in [n+t+1]$ is only polylogarithmic in $(n+t+1)$. When invoking **batchMAC.Auth** to transmit the MAC tags σ to P , it also costs V $\text{poly}(\log m, \lambda)$ time and memory. Within Step 2 and Step 3, which are identical to QuickSilver, V only needs $O(M)$ memory instead of $O(m)$. Overall, the verifier only needs $\max\{O(M), \text{poly}(\log m)\}$ memory. Check Construction 9 and Appendix B for details on the computation and memory efficiency.

Remark 2. We remark that we only propose a new roadmap towards constructing lattice-based ZK with succinct proof size, verification time, and memory, not a complete AoK. The reason is that we did not prove the key privacy of our batch-MAC schemes against a malicious user (or prover in this setting). However, one may combine our AoK framework with an opening argument for linear hash functions to build a complete circuit AoK. In this case, one can also view our BatchMAC-QuickSilver framework as an efficient heuristic compiler that reduces an AoK for general circuit satisfiability to an AoK for linear hash openings.

Acknowledgement: The authors thank the TCC 2026 reviewers for their insightful comments. In an early version of the paper, one of the main constructions was based on pairings. A TCC reviewer suggested that pairings can be replaced by the Paillier groups.

References

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. In: Dodis and Shrimpton [37], pp. 421–452. https://doi.org/10.1007/978-3-031-15985-5_15
2. Abram, D., Malavolta, G., Roy, L.: Succinct oblivious tensor evaluation and applications: Adaptively-secure laconic function evaluation and trapdoor hashing for all circuits. In: Koutecký, M., Bansal, N. (eds.) 57th ACM STOC. pp. 1875–1886. ACM Press (Jun 2025). <https://doi.org/10.1145/3717823.3718164>
3. Abram, D., Malavolta, G., Roy, L.: How to authenticate a non-deterministic computation. Cryptology ePrint Archive, Paper 2026/741 (2026), <https://eprint.iacr.org/2026/741>
4. Abram, D., Roy, L., Scholl, P.: Succinct homomorphic secret sharing. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VI. LNCS, vol. 14656, pp. 301–330. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58751-1_11

5. Agarwal, A., Baum, C., Braun, L., Scholl, P.: Low-bandwidth mixed arithmetic in VOLE-based ZK from low-degree PRGs. In: Fehr, S., Fouque, P.A. (eds.) EU-ROCRYPT 2025, Part IV. LNCS, vol. 15604, pp. 396–426. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91134-7_14
6. Agarwal, S., Jain, A., Srinivasan, A., Wu, D.J.: Simultaneous-message and succinct secure computation: Reusable and multiparty protocols. In: Daemen, J., Thomé, E. (eds.) Advances in Cryptology – EUROCRYPT 2026. pp. 94–123. Springer Nature Switzerland, Cham (2026)
7. Applebaum, B., Beimel, A., Ishai, Y., Kushilevitz, E., Liu, T., Vaikuntanathan, V.: Succinct computational secret sharing. In: Saha, B., Servedio, R.A. (eds.) 55th ACM STOC. pp. 1553–1566. ACM Press (Jun 2023). <https://doi.org/10.1145/3564246.3585127>
8. Applebaum, B., Ishai, Y., Kushilevitz, E., Waters, B.: Encoding functions with constant online rate or how to compress garbled circuits keys. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 166–184. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1_10
9. Attrapadung, N., Matsuda, T., Nishimaki, R., Yamada, S., Yamakawa, T.: Constrained PRFs for NC^1 in traditional groups. In: Shacham and Boldyreva [65], pp. 543–574. https://doi.org/10.1007/978-3-319-96881-0_19
10. Baum, C., Beullens, W., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: Shorter, tighter, FAESTer: Optimizations and improved (QROM) analysis for VOLE-in-the-head signatures. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part VI. LNCS, vol. 16005, pp. 124–156. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01887-8_5
11. Baum, C., Braun, L., Delpech de Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 581–615. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_19
12. Baum, C., Braun, L., Munch-Hansen, A., Scholl, P.: Moz $\mathbb{Z}_2^k$ arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_2^k$. In: Dodis and Shrimpton [37], pp. 329–358. https://doi.org/10.1007/978-3-031-15985-5_12
13. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac’n’cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 92–122. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_4
14. Behera, A.R., Meyer, P., Orlandi, C., Roy, L., Scholl, P.: Privately constrained PRFs from DCR: Puncturing and bounded waring rank. In: Applebaum, B., Lin, H.R. (eds.) TCC 2025, Part I. LNCS, vol. 16268, pp. 301–332. Springer, Cham (Dec 2025). https://doi.org/10.1007/978-3-032-12287-2_11
15. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 410–428. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_23
16. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer, Berlin, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42045-0_15

17. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press (Oct 2018). <https://doi.org/10.1145/3243734.3243868>
18. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Resch, N., Scholl, P.: Correlated pseudorandomness from expand-accumulate codes. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 603–633. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_21
19. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: Cavallaro et al. [27], pp. 291–308. <https://doi.org/10.1145/3319535.3354255>
20. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 489–518. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_16
21. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer, Berlin, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0_29
22. Boyle, E., Jain, A., Servan-Schreiber, S., Srinivasan, A.: Simultaneous-message and succinct secure computation. In: Fehr and Fouque [38], pp. 207–239. https://doi.org/10.1007/978-3-031-91092-0_8
23. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 3–33. Springer, Cham (May 2019). https://doi.org/10.1007/978-3-030-17656-3_1
24. Brakerski, Z., Tsabary, R., Vaikuntanathan, V., Wee, H.: Private constrained PRFs (and more) from LWE. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 264–302. Springer, Cham (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2_10
25. Brakerski, Z., Vaikuntanathan, V.: Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 1–30. Springer, Berlin, Heidelberg (Mar 2015). https://doi.org/10.1007/978-3-662-46497-7_1
26. Canetti, R., Chen, Y.: Constraint-hiding constrained PRFs for NC^1 from LWE. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 446–476. Springer, Cham (Apr / May 2017). https://doi.org/10.1007/978-3-319-56620-7_16
27. Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.): ACM CCS 2019. ACM Press (Nov 2019)
28. Chen, Y., Vaikuntanathan, V., Wee, H.: GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In: Shacham and Boldyreva [65], pp. 577–607. https://doi.org/10.1007/978-3-319-96881-0_20
29. Yu Chiang, J.H., Damgård, I., Duro, W.R., Engan, S., Kolby, S., Scholl, P.: Post-quantum threshold ring signature applications from VOLE-in-the-head. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 4664–4678. ACM Press (Oct 2025). <https://doi.org/10.1145/3719027.3744854>
30. Couteau, G., Hegde, A., Pu, S.: Enhanced trapdoor hashing from DDH and DCR. In: Fehr and Fouque [39], pp. 123–151. https://doi.org/10.1007/978-3-031-91095-1_5

31. Couteau, G., Meyer, P., Passelègue, A., Riahinia, M.: Constrained pseudorandom functions from homomorphic secret sharing. In: Hazay, C., Stam, M. (eds.) EU-ROCRYPT 2023, Part III. LNCS, vol. 14006, pp. 194–224. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30620-4_7
32. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In: Malkin and Peikert [53], pp. 502–534. https://doi.org/10.1007/978-3-030-84252-9_17
33. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Berlin, Heidelberg (Feb 2001). https://doi.org/10.1007/3-540-44586-2_9
34. Dao, Q., Ishai, Y., Jain, A., Lin, H.: Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 315–348. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38545-2_11
35. Dietz, M., Li, H., Lin, H.: TinyLabels: How to compress garbled circuit input labels, efficiently. In: Fehr and Fouque [39], pp. 245–274. https://doi.org/10.1007/978-3-031-91095-1_9
36. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-point zero knowledge and its applications. In: Tessaro, S. (ed.) ITC 2021. LIPIcs, vol. 199, pp. 5:1–5:24. Schloss Dagstuhl (Jul 2021). <https://doi.org/10.4230/LIPIcs.ITC.2021.5>
37. Dodis, Y., Shrimpton, T. (eds.): CRYPTO 2022, Part IV, LNCS, vol. 13510. Springer, Cham (Aug 2022)
38. Fehr, S., Fouque, P.A. (eds.): EUROCRYPT 2025, Part V, LNCS, vol. 15605. Springer, Cham (May 2025)
39. Fehr, S., Fouque, P.A. (eds.): EUROCRYPT 2025, Part VI, LNCS, vol. 15606. Springer, Cham (May 2025)
40. Garg, S., Ostrovsky, R., Srinivasan, A.: Adaptive garbled RAM from laconic oblivious transfer. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 515–544. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_18
41. Garg, S., Srinivasan, A.: Adaptively secure garbling with near optimal online complexity. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 535–565. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_18
42. Hsieh, Y.C., Lin, H., Luo, J.: Attribute-based encryption for circuits of unbounded depth from lattices. In: 64th FOCS. pp. 415–434. IEEE Computer Society Press (Nov 2023). <https://doi.org/10.1109/FOCS57990.2023.00031>
43. Ishai, Y., Li, H., Lin, H.: Succinct homomorphic MACs from groups and applications. In: 66th FOCS. pp. 2505–2545. IEEE Computer Society Press (Dec 2025). <https://doi.org/10.1109/FOCS63196.2025.00131>
44. Ishai, Y., Li, H., Lin, H.: A unified framework for succinct garbling from homomorphic secret sharing. In: Kalai and Kamara [45], pp. 390–425. https://doi.org/10.1007/978-3-032-01884-7_13
45. Kalai, Y.T., Kamara, S.F. (eds.): CRYPTO 2025, Part IV, LNCS, vol. 16003. Springer, Cham (Aug 2025)
46. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 669–684. ACM Press (Nov 2013). <https://doi.org/10.1145/2508859.2516668>

47. Kim, J., Li, H., Lin, H., Liu, Z.: Simd hss and ahmac from interval encoding with application to one-bit-per-gate garbling. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology – CRYPTO 2026*. pp. 658–691. Springer Nature Switzerland, Cham (2026)
48. Kolesnikov, V., Pecený, S., Raghuraman, S., Rindal, P.: Stationary syndrome decoding for improved PCGs. In: Kalai, Y.T., Kamara, S.F. (eds.) *CRYPTO 2025*, Part I. LNCS, vol. 16000, pp. 284–317. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01855-7_10
49. Li, H., Lin, H., Lu, G.: Succinct garbled circuits with low-depth garbling algorithms. In: Daemen, J., Thomé, E. (eds.) *Advances in Cryptology – EUROCRYPT 2026*. pp. 246–275. Springer Nature Switzerland, Cham (2026)
50. Li, Z., Xing, C., Yao, Y., Yuan, C.: Efficient pseudorandom correlation generators for any finite field. In: Fehr and Fouque [38], pp. 145–175. https://doi.org/10.1007/978-3-031-91092-0_6
51. Liu, H., Wang, X., Yang, K., Yu, Y.: BitGC: Garbled circuits with 1 bit per gate. In: Fehr, S., Fouque, P.A. (eds.) *EUROCRYPT 2025*, Part VII. LNCS, vol. 15607, pp. 437–466. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91098-2_16
52. Liu, L., Liu, T., Wei, L.: Pairing-based succinct half-chosen VOLE. *Cryptology ePrint Archive*, Paper 2026/1922 (2026), <https://eprint.iacr.org/2026/1922>
53. Malkin, T., Peikert, C. (eds.): *CRYPTO 2021*, Part III, LNCS, vol. 12827. Springer, Cham, Virtual Event (Aug 2021)
54. Meyer, P., Orlandi, C., Roy, L., Scholl, P.: Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. In: Kalai and Kamara [45], pp. 426–458. https://doi.org/10.1007/978-3-032-01884-7_14
55. Meyer, P., Orlandi, C., Roy, L., Scholl, P.: Computationally succinct authentication from dcr. In: Daemen, J., Thomé, E. (eds.) *Advances in Cryptology – EUROCRYPT 2026*. pp. 314–344. Springer Nature Switzerland, Cham (2026)
56. Ng, L.K.L., Rindal, P., Shah, A.: LogVOLE: Succinct and efficient chosen-input VOLE for ZK and beyond. *Cryptology ePrint Archive*, Paper 2026/925 (2026), <https://eprint.iacr.org/2026/925>
57. Orlandi, C., Scholl, P., Yakoubov, S.: The rise of paillier: Homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.X. (eds.) *EUROCRYPT 2021*, Part I. LNCS, vol. 12696, pp. 678–708. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_24
58. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) *EUROCRYPT’99*. LNCS, vol. 1592, pp. 223–238. Springer, Berlin, Heidelberg (May 1999). https://doi.org/10.1007/3-540-48910-X_16
59. Peikert, C., Shiehian, S.: Privately constraining and programming PRFs, the LWE way. In: Abdalla, M., Dahab, R. (eds.) *PKC 2018*, Part II. LNCS, vol. 10770, pp. 675–701. Springer, Cham (Mar 2018). https://doi.org/10.1007/978-3-319-76581-5_23
60. Quach, W., Wee, H., Wichs, D.: Laconic function evaluation and applications. In: Thorup, M. (ed.) *59th FOCS*. pp. 859–870. IEEE Computer Society Press (Oct 2018). <https://doi.org/10.1109/FOCS.2018.00086>
61. Raghuraman, S., Rindal, P., Tanguy, T.: Expand-convolute codes for pseudorandom correlation generators from LPN. In: Handschuh, H., Lysyanskaya, A. (eds.) *CRYPTO 2023*, Part IV. LNCS, vol. 14084, pp. 602–632. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_19

62. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). <https://doi.org/10.1145/1060590.1060603>
63. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin and Peikert [53], pp. 687–717. https://doi.org/10.1007/978-3-030-84252-9_23
64. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed vector-OLE: Improved constructions and implementation. In: Cavallaro et al. [27], pp. 1055–1072. <https://doi.org/10.1145/3319535.3363228>
65. Shacham, H., Boldyreva, A. (eds.): CRYPTO 2018, Part II, LNCS, vol. 10992. Springer, Cham (Aug 2018)
66. Wee, H.: Circuit ABE with $\text{poly}(\text{depth}, \lambda)$ -sized ciphertexts and keys from lattices. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part III. LNCS, vol. 14922, pp. 178–209. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68382-4_6
67. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In: 2021 IEEE Symposium on Security and Privacy. pp. 1074–1091. IEEE Computer Society Press (May 2021). <https://doi.org/10.1109/SP40001.2021.00056>
68. Weng, C., Yang, K., Yang, Z., Xie, X., Wang, X.: AntMan: Interactive zero-knowledge proofs with sublinear communication. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 2901–2914. ACM Press (Nov 2022). <https://doi.org/10.1145/3548606.3560667>
69. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2986–3001. ACM Press (Nov 2021). <https://doi.org/10.1145/3460120.3484556>
70. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1607–1626. ACM Press (Nov 2020). <https://doi.org/10.1145/3372297.3417276>
71. Yang, Y.: Justvengers: Batched VOLE ZK disjunctions in $\mathcal{O}(R + B + C)$ communication. In: Applebaum, B., Lin, H.R. (eds.) TCC 2025, Part IV. LNCS, vol. 16271, pp. 105–137. Springer, Cham (Dec 2025). https://doi.org/10.1007/978-3-032-12290-2_4
72. Yang, Y., Heath, D., Hazay, C., Kolesnikov, V., Venkitasubramaniam, M.: Batchman and robin: Batched and non-batched branching for interactive ZK. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 1452–1466. ACM Press (Nov 2023). <https://doi.org/10.1145/3576915.3623169>

A Proof of Lemma 4

In this section, we formally prove Lemma 4. We prove it by the hybrid argument. Consider the following $n + 1$ hybrid distributions.

– Hybrid₀:

$$\left\{ \begin{array}{l} \text{pp} \\ t_1, \dots, t_n \\ \tau_1, \dots, \tau_n \\ t_1^w, \dots, t_n^w \end{array} \middle| \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [n] : (t_j, \tau_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ \mathbf{w} \xleftarrow{\$} [\ell]^n \end{array} \right\}$$

– Hybrid_i:

$$\left\{ \begin{array}{c|l} \begin{array}{c} \text{pp} \\ t_1, \dots, t_n \\ \tau_1, \dots, \tau_n \\ f^{\mu_{1,1}} \cdot t_1^{w_1}, \dots, f^{\mu_{n,1}} \cdot t_n^{w_1} \\ \dots \\ f^{\mu_{1,i}} \cdot t_1^{w_i}, \dots, f^{\mu_{n,i}} \cdot t_n^{w_i} \\ t_1^{w_{i+1}}, \dots, t_n^{w_{i+1}} \\ \dots \\ t_1^{w_n}, \dots, t_n^{w_n} \end{array} & \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [n] : (t_j, \tau_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ \mathbf{w} \xleftarrow{\$} [\ell]^n, \mathbf{w} = (w_1, \dots, w_n) \\ \forall j \in [n], \forall k \in [i] : \mu_{j,k} \xleftarrow{\$} \mathbb{Z}_N \end{array} \end{array} \right\}.$$

– Hybrid_{i+1}:

$$\left\{ \begin{array}{c|l} \begin{array}{c} \text{pp} \\ t_1, \dots, t_n \\ \tau_1, \dots, \tau_n \\ f^{\mu_{1,1}} \cdot t_1^{w_1}, \dots, f^{\mu_{n,1}} \cdot t_n^{w_1} \\ \dots \\ f^{\mu_{1,i}} \cdot t_1^{w_i}, \dots, f^{\mu_{n,i}} \cdot t_n^{w_i} \\ f^{\mu_{1,i+1}} \cdot t_1^{w_{i+1}}, \dots, f^{\mu_{n,i+1}} \cdot t_n^{w_{i+1}} \\ \dots \\ t_1^{w_n}, \dots, t_n^{w_n} \end{array} & \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [n] : (t_j, \tau_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ \mathbf{w} \xleftarrow{\$} [\ell]^n, \mathbf{w} = (w_1, \dots, w_n) \\ \forall j \in [n], \forall k \in [i+1] : \mu_{j,k} \xleftarrow{\$} \mathbb{Z}_N \end{array} \end{array} \right\}.$$

– Hybrid_n:

$$\left\{ \begin{array}{c|l} \begin{array}{c} \text{pp} \\ t_1, \dots, t_n \\ \tau_1, \dots, \tau_n \\ f^{\mu_1} \cdot t_1^{\mathbf{w}}, \dots, f^{\mu_n} \cdot t_n^{\mathbf{w}} \end{array} & \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [n] : (t_j, \tau_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ \mathbf{w} \xleftarrow{\$} [\ell]^n \\ \forall j \in [n] : \mu_j \xleftarrow{\$} \mathbb{Z}_N^n \end{array} \end{array} \right\}.$$

It remains to show that Hybrid_i and Hybrid_{i+1} are computationally indistinguishable, for all $i = 0, 1, \dots, n-1$. Additionally give the distinguisher $w_{i+2}, \dots, w_n$ and $\mu_{1,k}, \dots, \mu_{n,k}$ for all $k \in [i]$, which only strengthens the security guarantee. The distinguishing game is reduced to distinguishing

$$\left\{ \begin{array}{c|l} \begin{array}{c} \text{pp} \\ t_1, \dots, t_n \\ \tau_1, \dots, \tau_n \\ t_1^{w_{i+1}}, \dots, t_n^{w_{i+1}} \end{array} & \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [n] : (t_j, \tau_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ w_{i+1} \xleftarrow{\$} [\ell] \end{array} \end{array} \right\},$$

and

$$\left\{ \begin{array}{c|l} \begin{array}{c} \text{pp} \\ t_1, \dots, t_n \\ \tau_1, \dots, \tau_n \\ f^{\mu_{1,i+1}} \cdot t_1^{w_{i+1}}, \dots, f^{\mu_{n,i+1}} \cdot t_n^{w_{i+1}} \end{array} & \begin{array}{l} \text{pp} := (G, F, H, f, N, \ell) \xleftarrow{\$} \text{Gen}_{\text{Paillier}}(1^\lambda) \\ \forall j \in [n] : (t_j, \tau_j) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}) \\ w_{i+1} \xleftarrow{\$} [\ell] \\ \forall j \in [n] : \mu_{j,i+1} \xleftarrow{\$} \mathbb{Z}_N \end{array} \end{array} \right\}.$$

Their indistinguishability follows directly from the $(n-1)$ -ary enhanced DDH assumption in the Paillier group G . Thus, we have reduced the lemma to the DCR assumption.

B Modified Construction of NI-OTE

In this section, we show a slight modification of [2]’s NI-OTE scheme that possesses statistical hasher privacy. The construction of [2] achieved poly-logarithmic communication by induction. For $|\mathbf{x}| = m = t^r \cdot n$, $n = k \log q$, $k = \Theta(\lambda)$, the hasher compresses $\mathbf{x}$ by t in each iteration, by splitting it into pieces of size tn and multiplying each of them by $\mathbf{A}$. The overall hash function forms a Merkle tree, composed of linear hashes. Intuitively, the encoder’s messages enable the shares to climb up the tree, from the root to the leaves. We first review [2]’s subroutine hOTE, which formally describes a single iteration.

Construction 11 (Subroutine hOTE of [2]) *Construct a half-succinct NI-OTE scheme hOTE = (Setup, Hash, Enc, HashEval, EncEval) for $\mathbb{Z}_q^{t \cdot n} \otimes \mathbb{Z}_q^n$ as follows.*

- Setup(1^λ): Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_2^{n \times tn}$ and $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_2^{n \times (tn \cdot n \cdot n)}$, then return $\text{pk} := (\mathbf{A}, \mathbf{B})$.
- Hash(pk, $\mathbf{x}$): Compute $\mathbf{d} := \mathbf{A} \cdot \mathbf{x}$ and return $d := \mathbf{d}$ and $\psi := \mathbf{x}$.
- Enc(pk, $\mathbf{y}$): Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\hat{\mathbf{E}} \xleftarrow{\$} \chi(\lambda)$, and $\mathbf{E}' \xleftarrow{\$} \chi(\lambda)$. Compute

$$\mathbf{C} := \mathbf{A}^\top \cdot \mathbf{B} \cdot (\mathbf{I}_{tn \cdot n} \otimes \mathbf{s} \otimes \mathbf{g}_q^\top) + \mathbf{A}^\top \cdot \mathbf{E}' + \hat{\mathbf{E}} + \mathbf{I}_{tn} \otimes \mathbf{y}^\top.$$

Return $E := \mathbf{C}$ and $\phi := \mathbf{s}$.

- HashEval(pk, E, ψ): Return $\mathbf{v} := \mathbf{C}^\top \cdot \mathbf{x}$.
- EncEval(pk, d, ϕ): Return $\mathbf{w} := -(\mathbf{I}_{tn \cdot n} \otimes \mathbf{s}^\top \otimes \mathbf{g}_q) \cdot \mathbf{B}^\top \cdot \mathbf{d}$.

There seems to be a dilemma in making hOTE hasher-private: because $\mathbf{A}^\top \cdot \mathbf{E}'$ counts into the error, $\mathbf{A}$ has to be low-norm; it’s impossible to achieve privacy through the linear hash function $\mathbf{A} \cdot \mathbf{x}$ with low-norm $\mathbf{A}$ in a large ring. We borrow a simple trick from [4], at the expense of increasing communication by $\text{poly}(\lambda)$ (which does not affect the asymptotic complexity).

Construction 12 (Subroutine phOTE with Hasher Privacy) *Construct a half-succinct NI-OTE scheme phOTE = (Setup, Hash, Enc, HashEval, EncEval) for $\mathbb{Z}_q^{t \cdot n} \otimes \mathbb{Z}_q^n$ as follows.*

- Setup(1^λ): Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_2^{n \times tn}$, $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_2^{n \times (tn \cdot n \cdot n)}$, and $\mathbf{H} \xleftarrow{\$} \mathbb{Z}_q^{n \times (n \log q + 2\lambda)}$. Return $\text{pk} := (\mathbf{A}, \mathbf{B}, \mathbf{H})$.
- Hash(pk, $\mathbf{x}$): First, sample a random vector $\mathbf{u} \xleftarrow{\$} \mathbb{Z}_2^{n \log q + 2\lambda}$. Compute $\mathbf{d} := \mathbf{A} \cdot \mathbf{x} + \mathbf{H} \cdot \mathbf{u}$. Return $d := \mathbf{d}$ and $\psi := (\mathbf{x}, \mathbf{u})$.
- Enc(pk, $\mathbf{y}$): Sample $\mathbf{S} \xleftarrow{\$} \mathbb{Z}_q^{n \times tn^2}$, $\hat{\mathbf{E}} \xleftarrow{\$} \chi(\lambda)$, and $\tilde{\mathbf{E}} \xleftarrow{\$} \chi(\lambda)$. Compute

$$\mathbf{C}_0 := \mathbf{A}^\top \cdot \mathbf{S} + \hat{\mathbf{E}} + \mathbf{I}_{tn} \otimes \mathbf{y}^\top,$$

$$\mathbf{C}_1 := \mathbf{H}^\top \cdot \mathbf{S} + \tilde{\mathbf{E}}.$$

Return $E := (\mathbf{C}_0, \mathbf{C}_1)$ and $\phi := \text{vec}(\mathbf{S})$ ³.

³ $\text{vec}(\mathbf{S})$ denotes the vectorisation of $\mathbf{S}$.

- $\text{HashEval}(\text{pk}, E, \psi)$: Return $\mathbf{v} := \mathbf{C}_0^\top \cdot \mathbf{x} + \mathbf{C}_1^\top \cdot \mathbf{u}$.
- $\text{EncEval}(\text{pk}, d, \phi)$: Return $\mathbf{w} := -\mathbf{S}^\top \cdot \mathbf{d}$.

Note that $\mathbf{H}$ is a random (public) matrix over $\mathbb{Z}_q$. Thus $\mathbf{u} \mapsto \mathbf{H}\mathbf{u}$ is a universal hash function. Thus, according to the leftover hash lemma, $\mathbf{H}\mathbf{u}$ is statistically close to a uniform random vector in $\mathbb{Z}_q^n$. Therefore, hasher privacy holds statistically. Encoder privacy follows directly from the LWE assumption. Correctness holds by calculating:

$$\begin{aligned} \mathbf{w} + \mathbf{v} &= \mathbf{C}_0^\top \cdot \mathbf{x} + \mathbf{C}_1^\top \cdot \mathbf{u} - \mathbf{S}^\top \cdot \mathbf{d} \\ &= \mathbf{S}^\top \mathbf{A} \mathbf{x} + \hat{\mathbf{E}}^\top \mathbf{x} + \mathbf{x} \otimes \mathbf{y} + \mathbf{S}^\top \mathbf{H} \mathbf{u} + \tilde{\mathbf{E}}^\top \mathbf{u} - \mathbf{S}^\top \mathbf{A} \mathbf{x} - \mathbf{S}^\top \mathbf{H} \mathbf{u} \\ &= \mathbf{x} \otimes \mathbf{y} + (\hat{\mathbf{E}}^\top \mathbf{x} + \tilde{\mathbf{E}}^\top \mathbf{u}). \end{aligned}$$

The error is bounded by

$$\|\hat{\mathbf{E}}^\top \mathbf{x} + \tilde{\mathbf{E}}^\top \mathbf{u}\|_\infty \leq (tn + n \log q + 2\lambda) \cdot E(\lambda) \cdot \|\mathbf{x}\|_\infty \leq tn(n+1) \cdot E(\lambda) \cdot \|\mathbf{x}\|_\infty.$$

Recall that in hOTE, the error bound is $tn(n+1) \cdot E(\lambda) \cdot \|\mathbf{x}\|_\infty$. Thus, our hasher-private modification phOTE does not increase the error magnitude. Moreover, the encoder's evaluation algorithm, namely $-\mathbf{S}^\top \mathbf{d}$, is still a linear function on $(\mathbf{d} \otimes \phi \otimes \mathbf{g}_q^\top) = (\mathbf{d} \otimes \text{vec}(\mathbf{S}) \otimes \mathbf{g}_q^\top)$. We denote this public decoding matrix derived from pk as $\hat{\mathbf{P}}$.

However, the norm of the digest $\mathbf{d}$ increases significantly, which will consequently increase the overall error if we alter hOTE to phOTE in each iteration of OTE. The fix is simple: only use phOTE for the last iteration (the root of the Merkle tree).

Construction 13 (NI-OTE with Hasher Privacy) *Construct a fully-succinct NI-OTE scheme $\text{OTE} = (\text{Setup}, \text{Hash}, \text{Enc}, \text{HashEval}, \text{EncEval})$ for $\mathbb{Z}_q^m \otimes \mathbb{Z}_q^\ell$ as follows.*

- $\text{Setup}(1^\lambda)$: Return $\text{pk} \xleftarrow{\$} \text{phOTE.Setup}(1^\lambda)$.
- $\text{Hash}(\text{pk}, \mathbf{x})$: Set $\mathbf{x}_0 := \mathbf{x}$. Then for every $i \in [0..r-2]$ proceed as follows:
 - Parse $\mathbf{x}_i$ as the vertical concatenation of $(\mathbf{x}_{i,1}, \dots, \mathbf{x}_{i,t^{r-i-1}})$ where $\mathbf{x}_{i,j} \in \mathbb{Z}_q^{t \cdot n}$.
 - For every $j \in [t^{r-i-1}]$, compute

$$(\mathbf{d}_{i,j}, \psi_{i,j}) \xleftarrow{\$} \text{hOTE.Hash}(\text{pk}, \mathbf{x}_{i,j}).$$

- Define $\mathbf{x}_{i+1}$ to be the vertical concatenation of the $\mathbf{d}_{i,j}$. Additionally, compute

$$(\mathbf{d}_{r-1}, \psi_{r-1,1}) \xleftarrow{\$} \text{phOTE.Hash}(\text{pk}, \mathbf{x}_{r-1}).$$

Define $\mathbf{x}_r := \mathbf{d}_{r-1}$.

Return $d := \mathbf{x}_r$ and $\psi := \{\psi_{i,j}\}_{i \in [0..r-1], j \in [t^{r-i-1}]}$.

– $\text{hOTE.Enc}(\text{pk}, \mathbf{y})$: Set $\mathbf{y}_0 := \mathbf{y}$, then for $i \in [0..r-2]$, compute

$$(E_i, \phi_{i+1}) \stackrel{\$}{\leftarrow} \text{Enc}(\text{pk}, \mathbf{y}_i) \quad \text{and} \quad \mathbf{y}_{i+1} := \phi_{i+1} \otimes \mathbf{g}_q^\top.$$

Additionally, compute

$$(E_{r-1}, \phi_r) \stackrel{\$}{\leftarrow} \text{phOTE.Enc}(\text{pk}, \mathbf{y}_{r-1}).$$

Output $E := \{E_i\}_{i \in [0..r-1]}$ and $\phi := \phi_r$.

– $\text{HashEval}(\text{pk}, E, \psi)$: Let $\mathbf{P}$ be the public decoding matrix derived from pk in hOTE scheme. For all $i \in [0..r-2]$ and $j \in [t^{r-i-1}]$, compute

$$\mathbf{P}_i := (\mathbf{I}_{t^{r-i}} \otimes \mathbf{P}) \quad \text{and} \quad \mathbf{v}_{i,j} := \text{hOTE.HashEval}(\text{pk}, E_i, \psi_{i,j}).$$

Let $\mathbf{v}_i$ be the vertical concatenation of $\{\mathbf{v}_{i,j}\}_{j \in [t^{r-i-1}]}$. Additionally, compute

$$\mathbf{P}_{r-1} := (\mathbf{I}_t \otimes \hat{\mathbf{P}}) \quad \text{and} \quad \mathbf{v}_{r-1} := \text{phOTE.HashEval}(\text{pk}, E_{r-1}, \psi_{r-1,1}),$$

where $\hat{\mathbf{P}}$ is the public decoding matrix in phOTE scheme.

Return $\mathbf{v} := \sum_{i \in [0..r-1]} \left(\prod_{k=1}^i \mathbf{P}_k \right) \cdot \mathbf{v}_i$.

– $\text{EncEval}(\text{pk}, d, \phi)$: Let $\mathbf{P}_i$ be defined as above. Return

$$\mathbf{w} := \left(\prod_{i=1}^{r-1} \mathbf{P}_i \right) \cdot \text{phOTE.EncEval}(\text{pk}, d, \phi_r).$$

Remark 3. In the above construction, the randomness consumed within $\text{Hash}(\text{pk}, \mathbf{x})$ is exactly the randomness consumed within $\text{phOTE.Hash}(\text{pk}, \mathbf{x}_{r-1})$. Note that the hash randomness in Construction 12 is the vector $\mathbf{u}$, which is of size $\text{poly}(\lambda)$. Thus, the hash randomness size of the OTE in Construction 13 is $\text{poly}(\lambda)$.

C Constrained PRF with Succinct Key

A constrained PRF is a pseudorandom function supporting constrained keys for evaluating the function over restricted input sets. As an interesting application, our succinct batch-MAC scheme can be used to construct a constrained PRF for general circuit constraints, with the constrained key's size sublinear in the circuit size.

We first formally define a constrained PRF for general circuit constraints with 1-key selective security.

Definition 11 (Constrained Pseudorandom Functions). Denote by λ a security parameter. A Constrained Pseudorandom Function (CPRF) with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$, key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda \in \mathbb{N}}$, and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$, that supports a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where circuits in $\mathcal{C}_\lambda$ has domain $\mathcal{X}_\lambda$ and range $\{0, 1\}$, consists of the following four algorithms:

- $\text{KeyGen}(1^\lambda) \rightarrow (\text{pp}, \text{msk})$: On input the security parameter λ , the master key generation algorithm outputs a public parameter pp and a master secret key $\text{msk} \in \mathcal{K}$.
- $\text{Eval}(\text{pp}, \text{msk}, x) \rightarrow y$: On input the public parameter pp , the master secret key msk , and an input $x \in \mathcal{X}$, the evaluation algorithm outputs a value $y \in \mathcal{Y}$.
- $\text{Constrain}(\text{pp}, \text{msk}, C \in \mathcal{C}) \rightarrow \text{ck}_C$: On input the public parameter pp , the master secret key msk , and a circuit $C \in \mathcal{C}$, the constrained key generation algorithm outputs a constrained key ck_C .
- $\text{CEval}(\text{pp}, \text{ck}_C, C, x) \rightarrow y$: On input the public parameter pp , a constrained key ck_C , the constrained circuit C , and an input $x \in \mathcal{X}$, the constrained evaluation algorithm outputs a value $y \in \mathcal{Y}$.

For a constrained PRF, we require the following correctness and security properties to hold:

Correctness: For any security parameter λ , any constraint $C \in \mathcal{C}$, and any input $x \in \mathcal{X}$ such that $C(x) = 0$, we have:

$$\Pr \left[\begin{array}{l} \text{Eval}(\text{pp}, \text{msk}, x) \\ \neq \text{CEval}(\text{pp}, \text{ck}_C, C, x) \end{array} : \begin{array}{l} \text{msk} \leftarrow \text{KeyGen}(\text{pp}) \\ \text{ck}_C \leftarrow \text{Constrain}(\text{pp}, \text{msk}, C) \end{array} \right] \leq \text{negl}(\lambda).$$

1-Key Selective Security: For any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$:

$$\left| \Pr[\text{Exp}_{\text{CPRF}}^{\mathcal{A},0}(\lambda) = 1] - \Pr[\text{Exp}_{\text{CPRF}}^{\mathcal{A},1}(\lambda) = 1] \right| \leq \text{negl}(\lambda),$$

where the experiment $\text{Exp}_{\text{CPRF}}^{\mathcal{A},b}$ is as follows:

1. Launch $\mathcal{A}(1^\lambda)$ and receives a selective challenge constraint $C \in \mathcal{C}_\lambda$.
2. Run $(\text{pp}, \text{msk}) \leftarrow \text{KeyGen}(1^\lambda)$, $\text{sk}_C \leftarrow \text{Constrain}(\text{pp}, \text{msk}, C)$, and send pp, sk_C to the adversary $\mathcal{A}$.
3. Answer queries x from $\mathcal{A}$ with evaluations $y \leftarrow \text{Eval}(\text{pp}, \text{msk}, x)$.
4. Receive from $\mathcal{A}$ a challenge x^* that's never queried before and such that $C(x^*) \neq 0$. Sends y^* to $\mathcal{A}$ computed as follows:

$$\begin{array}{ll} y^* \leftarrow \text{Eval}(\text{pp}, \text{msk}, x^*) & \text{if } b = 0 \\ y^* \xleftarrow{\$} \mathcal{Y}_\lambda & \text{if } b = 1. \end{array}$$

5. Answer queries $x \neq x^*$ from $\mathcal{A}$ as in step 3.
6. In the end, $\mathcal{A}$ outputs a bit b' as the experiment result.

Our construction of CPRF from batch-MAC follows from recent progress in homomorphic secret sharing (HSS) that IT-MACs ($\sigma = \mathbf{x} \cdot \Delta + \mathbf{k}$) can (1) do homomorphic evaluations on themselves; (2) be homomorphically multiplied with HSS encryptions. These techniques are named algebraic homomorphic MACs (aHMACs) [43] and extended HSS evaluation [31], respectively. These two schemes can both be constructed from a variety of assumptions, as shown

in [43], including the circular power-DDH assumption over prime-order or Pailier groups, the KDM-DCR assumption, and the circular power-ring-LWE assumption. We summarize these ring-LWE-based constructions in the following lemmas.

Lemma 6 (Algebraic Homomorphic MACs under Lattices [43]). *Let $B < 2^{\text{poly}(\lambda)}$ be a bound on input values, $\mathcal{R}$ be the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where $n(\lambda)$ is a power-of-two, $p \geq B \cdot \lambda^{\omega(1)}$, $q = p \cdot \Delta$ be two moduli, where $\Delta > B^2 \cdot \lambda^{\omega(1)}$ is a scaling factor, namely $B \ll p \ll q$. Let $\mathcal{D}_{\text{sk}}(\lambda), \mathcal{D}_{\text{err}}(\lambda) \in \mathcal{R}$ be error and secret distributions with coefficients bounded by $\text{poly}(\lambda)$. Let $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ be a PRG seed.*

Assuming the Circular Power Ring-LWE (CP-RLWE) assumption, there exists an algebraic homomorphic MAC scheme aHMAC for arbitrary polynomial-sized circuit, within which

- the global secret key is $s \xleftarrow{\$} \mathcal{D}_{\text{sk}}(\lambda)$,
- for input $x \leq B$, the MAC key is $k_x \in \mathcal{R}_p$, the MAC tag is $\sigma_x := k_x + s \cdot x$ computed in $\mathcal{R}$

Further, aHMAC consists of three efficient algorithms:

- $\text{KGen}(1^\lambda, s \in \mathcal{R})$ takes as input the global secret key s , and outputs a secret key sk and an evaluation key evk .
- $\text{EvalTag}(\text{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$ takes as inputs the evaluation key evk , a circuit $C : \mathbb{Z}_B^{\ell_x} \rightarrow \mathbb{Z}_B^{\ell_z}$, input values $\mathbf{x} = (x_1, \dots, x_{\ell_x}) \in \mathbb{Z}_B^{\ell_x}$, and their associated tags $\sigma_{\mathbf{x}} = (\sigma_{x_1}, \dots, \sigma_{x_{\ell_x}})$. It outputs tags $\sigma_{\mathbf{z}} = (\sigma_{z_1}, \dots, \sigma_{z_{\ell_z}}) \in \mathcal{R}^{\ell_z}$ authenticating $\mathbf{z} := C(\mathbf{x})$.
- $\text{EvalKey}(\text{sk}, C, k_{\mathbf{x}})$ takes as inputs the secret key sk , a circuit $C : \mathbb{Z}_B^{\ell_x} \rightarrow \mathbb{Z}_B^{\ell_z}$ and the input keys $k_{\mathbf{x}} = (k_{x_1}, \dots, k_{x_{\ell_x}})$. It outputs MAC keys $k_{\mathbf{z}} = (k_{z_1}, \dots, k_{z_{\ell_z}}) \in \mathcal{R}^{\ell_z}$ for $\mathbf{z} := C(\mathbf{x})$.

Specifically, the evaluation key evk is formulated as

$$\begin{aligned} \text{pp} &= (\mathcal{R}, p, q, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}}), \text{seed} \xleftarrow{\$} \{0, 1\}^\lambda, a \xleftarrow{\$} \mathcal{R}_q, e_1, e_2 \xleftarrow{\$} \mathcal{D}_{\text{err}}, \\ \text{evk} &:= (\text{pp}, \text{seed}, a, s \cdot a + e_1, s^2 \cdot a + e_2 - s \cdot \Delta). \end{aligned}$$

The aHMAC scheme satisfies correctness, namely for all $\lambda \in \mathbb{N}$, and any input $\mathbf{x}$:

$$\Pr \left[\begin{array}{c} \sigma_{\mathbf{x}} = k_{\mathbf{x}} + s \cdot \mathbf{x} \\ (\text{evk}, \text{sk}) \xleftarrow{\$} \text{KGen}(1^\lambda, s) \\ \mathbf{z} = C(\mathbf{x}) \\ \sigma_{\mathbf{z}} \leftarrow \text{EvalTag}(\text{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}) \\ k_{\mathbf{z}} \leftarrow \text{EvalKey}(\text{sk}, C, k_{\mathbf{x}}) \end{array} \middle| \begin{array}{c} \sigma_{\mathbf{z}} \neq k_{\mathbf{z}} + s \cdot \mathbf{z} \\ (\text{over } \mathcal{R}) \end{array} \right] \leq \text{negl}(\lambda)$$

Lemma 7 (Extended HSS Evaluation under Lattices [23, 31]). *Set $B, \mathcal{R}, p, q, \Delta, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}}$ as in Lemma 6. Let $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ be a PRG seed, and define*

$\text{pp} := (\mathcal{R}, p, q, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}})$. Assuming the RLWE assumption, there exists a pair of efficient deterministic algorithms, ExtEval_0 and ExtEval_1 , such that: for every logarithmic function $d(\lambda)$, every polynomial $p(\lambda)$, there exists a negligible function $\text{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, Boolean circuit $C : \{0, 1\}^{\ell_y} \rightarrow \{0, 1\}$ with $|C| < p(\lambda)$, $\text{Depth}(C) < d(\lambda)$ ⁴, global secret key $s \in \mathcal{D}_{\text{sk}}$, inputs $x \in \mathbb{Z}_B$ and $\mathbf{y} \in \{0, 1\}^{\ell_y}$, and additive shares (over $\mathcal{R}$) $\langle s \cdot x \rangle_1 - \langle s \cdot x \rangle_0 = sx$, $\langle x \rangle_1 - \langle x \rangle_0 = x$, the following holds:

$$\Pr \left[\begin{array}{l} w_1^z = w_0^z + s \cdot z \\ v_1^z = v_0^z + z \\ \text{(over } \mathcal{R}) \end{array} \middle| \begin{array}{l} (w_b^z, v_b^z) \leftarrow \text{ExtEval}_b(\text{pp}, \text{seed}, \text{ct}_{\mathbf{y}}, C, \langle s \cdot x \rangle_b, \langle x \rangle_b) \\ z := x \cdot C(\mathbf{y}) \text{ (over } \mathbb{Z}) \end{array} \right] > 1 - \text{negl}(\lambda),$$

over the randomness of seed and $\text{ct}_{\mathbf{y}}$. Further, the ciphertext $\text{ct}_{\mathbf{y}}$ is computed (over $\mathcal{R}_q$) as follows:

$$\begin{aligned} \mathbf{a}' &\xleftarrow{\$} \mathcal{R}_q^{\ell_y}, \quad r_1, r_2 \xleftarrow{\$} \mathcal{D}_{\text{sk}}, \quad \mathbf{e}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}'_1, \mathbf{e}'_2 \xleftarrow{\$} \mathcal{D}_{\text{err}}^{\ell_y}, \quad \mathbf{b}' := s \cdot \mathbf{a}' + \mathbf{e}, \\ \mathbf{c}_1 &:= r_1 \cdot \mathbf{a}' + \mathbf{e}_1 + \mathbf{y} \cdot \Delta, \quad \mathbf{c}'_1 := r_1 \cdot \mathbf{b}' + \mathbf{e}'_1, \\ \mathbf{c}_2 &:= r_2 \cdot \mathbf{a}' + \mathbf{e}_2, \quad \mathbf{c}'_2 := r_2 \cdot \mathbf{b}' + \mathbf{e}'_2 + \mathbf{y} \cdot \Delta. \\ \text{ct}_{\mathbf{y}} &:= (\mathbf{c}_1, \mathbf{c}'_1, \mathbf{c}_2, \mathbf{c}'_2). \end{aligned}$$

Now we are ready to construct a constrained PRF with sublinear keys. Our CPRF works for the class of polynomial-sized circuits $\mathcal{C} = \{\mathcal{C}_{\lambda}\}_{\lambda}$ where

$$\mathcal{C}_{\lambda} := \{C : \{0, 1\}^{\lambda} \rightarrow \{0, 1\} : |C| < \text{poly}(\lambda)\}.$$

We use ℓ_C to denote the length of C when described as a boolean vector. As shorthands, in the following we write $U_x(\cdot) = U(\cdot, x)$, and $\text{PRF}_x(\cdot) = \text{PRF}(\cdot, x)$, where U is a universal circuit such that $U_x(C) = C(x)$ for all circuits $C \in \mathcal{C}_{\lambda}$. Specifically, the domain of the CPRF is $\mathcal{X} := \{0, 1\}^{\lambda}$ and the range is $\mathcal{R}$.

Construction 14 (Constrained PRF) *Given the following ingredients:*

- a batch-MAC scheme $\text{batchMAC} = (\text{Setup}, \text{KeyGen}, \text{Hash}, \text{Auth}, \text{Ext})$, as defined in Definition 3, for message space $\mathbb{Z}_B^{\ell_C}$, where B is an upper bound for $\mathcal{D}_{\text{sk}}$'s coefficients;
- an aHMAC scheme $\text{aHMAC} = (\text{KGen}, \text{EvalTag}, \text{EvalKey})$ as shown in Lemma 6;
- a pair of extended HSS evaluation algorithms $(\text{ExtEval}_0, \text{ExtEval}_1)$ as shown in Lemma 7;
- a secure PRF $\text{PRF} : \mathcal{K} \times \{0, 1\}^{\lambda} \rightarrow \mathcal{R}$ with evaluation algorithm in NC1, and key space $\mathcal{K} = \{0, 1\}^{\ell_k}$,

we construct a constrained PRF $\text{CPRF} = (\text{KeyGen}, \text{Eval}, \text{Constrain}, \text{CEval})$ as follows:

- $\text{KeyGen}(1^{\lambda})$:

⁴ We use $\text{Depth}(C)$ to denote the depth of circuit C .

1. Set the public parameter $\text{pp}_{\text{HSS}} := (\mathcal{R}, p, q, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}})$ as in Lemma 6 and Lemma 7.
2. Sample a PRG seed $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ for aHMAC and HSS evaluation.
3. Compute $\text{pk}_{\text{batchMAC}} \xleftarrow{\$} \text{batchMAC.Setup}(1^\lambda)$.
4. Sample a PRF key for PRF, $k \xleftarrow{\$} \mathcal{K} = \{0, 1\}^{\ell_k}$.
5. Compute the HSS ciphertext for k as in Lemma 7, and denote it as ct_k .
6. Sample an HSS global secret key $s \xleftarrow{\$} \mathcal{D}_{\text{sk}}$.
7. Compute $(\text{sk}_{\text{aHMAC}}, \text{evk}_{\text{aHMAC}}) \xleftarrow{\$} \text{aHMAC.KGen}(1^\lambda, s)$.
8. Parse $s \in \mathcal{R}$ naturally as an n -dimension vector $\mathbf{s} = (s_1, \dots, s_n) \in \mathbb{Z}^n$ (recall that $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$).
For $i = 1, \dots, n$, compute MAC keys for C :

$$(\mathbf{k}_i, \text{st}_A^i) \xleftarrow{\$} \text{batchMAC.KeyGen}(\text{pk}_{\text{batchMAC}}).$$

9. Collect the randomness in all these n invocations of batchMAC.KeyGen algorithm and denote it as a vector $\mathbf{r}_{\text{MAC}}$.
10. Return

$$\text{pp} := (\text{pp}_{\text{HSS}}, \text{seed}, \text{pk}_{\text{batchMAC}}),$$

and

$$\text{msk} := (\text{ct}_k, \text{sk}_{\text{aHMAC}}, \text{evk}_{\text{aHMAC}}, \mathbf{r}_{\text{MAC}}).$$

– $\text{Eval}(\text{pp}, \text{msk}, x \in \{0, 1\}^\lambda)$:

1. For $i = 1, \dots, n$, recompute

$$(\mathbf{k}_i, \text{st}_A^i) \xleftarrow{\$} \text{batchMAC.KeyGen}(\text{pk}_{\text{batchMAC}}),$$

using randomness $\mathbf{r}_{\text{MAC}}$.

Concatenate $\mathbf{k}_i$ for $i = 1, \dots, n$ as an $\ell_C \times n$ matrix.

Parse the j -th row of this matrix as a vector $\mathbf{v}^j \in \mathbb{Z}^n$ for $j = 1, \dots, \ell_C$.

Parse each vector $\mathbf{v}^j$ naturally as a ring element in $\mathcal{R}$, and denote them altogether as $k_C \in \mathcal{R}^{\ell_C}$.

2. Compute

$$k_{C(x)} \leftarrow \text{aHMAC.EvalKey}(\text{sk}_{\text{aHMAC}}, U_x, k_C).$$

3. Compute

$$(w_0^z, v_0^z) \leftarrow \text{ExtEval}_0(\text{pp}_{\text{HSS}}, \text{seed}, \text{ct}_k, \text{PRF}_x, k_{C(x)}, 0).$$

4. Return v_0^z .

– $\text{Constrain}(\text{pp}, \text{msk}, C)$:

1. For $i = 1, \dots, n$, recompute

$$(\mathbf{k}_i, \text{st}_A^i) \xleftarrow{\$} \text{batchMAC.KeyGen}(\text{pk}_{\text{batchMAC}}),$$

using randomness $\mathbf{r}_{\text{MAC}}$.

2. Compute

$$(d_C, \text{st}_U) \xleftarrow{\$} \text{batchMAC.Hash}(\text{pk}_{\text{batchMAC}}, C)$$

and denote the randomness used in the algorithm as $\mathbf{r}_{\text{Hash}}$.

3. Compute compressive MACs for s_i :

$$e_s^i \xleftarrow{\$} \text{batchMAC.Auth}(\text{pk}_{\text{batchMAC}}, s_i, d_C, \text{st}_A^i)$$

for $i = 1, \dots, n$.

4. Return

$$\text{ck}_C := (\mathbf{r}_{\text{Hash}}, \text{ct}_k, \text{evk}_{\text{aHMAC}}, e_s^1, \dots, e_s^n).$$

– $\text{CEval}(\text{pp}, \text{ck}_C, C, x)$:

1. Recompute

$$(d_C, \text{st}_U) \xleftarrow{\$} \text{batchMAC.Hash}(\text{pk}_{\text{batchMAC}}, C)$$

using randomness $\mathbf{r}_{\text{Hash}}$.

2. For $i = 1, \dots, n$, compute the MAC tags for C :

$$\sigma_i \leftarrow \text{batchMAC.Ext}(\text{pk}_{\text{batchMAC}}, e_s^i, \text{st}_U).$$

Concatenate σ_i for $i = 1, \dots, n$ as an $\ell_C \times n$ matrix.

Parse the j -th row of this matrix as a vector $\mathbf{w}^j \in \mathbb{Z}^n$ for $j = 1, \dots, \ell_C$.

Parse each vector $\mathbf{w}^j$ naturally as a ring element in $\mathcal{R}$, and denote them altogether as $\sigma_C \in \mathcal{R}^{\ell_C}$.

3. Compute

$$\sigma_{C(x)} \leftarrow \text{aHMAC.EvalTag}(\text{evk}_{\text{aHMAC}}, U_x, C, \sigma_C).$$

4. Compute

$$(w_1^z, v_1^z) \leftarrow \text{ExtEval}_1(\text{pp}_{\text{HSS}}, \text{seed}, \text{ct}_k, \text{PRF}_x, \sigma_{C(x)}, C(x)).$$

5. Return v_1^z .

Analysis of key sizes. If we instantiate the underlying `batchMAC` with our LWE-based construction (i.e. Construction 9),

- the master secret key size will be $|\text{msk}| = |\text{ct}_k| + |\text{sk}_{\text{aHMAC}}| + |\text{evk}_{\text{aHMAC}}| + |\mathbf{r}_{\text{MAC}}| \leq \ell_k \cdot \text{poly}(\lambda) + \text{poly}(\lambda) + \text{poly}(\lambda) + \text{poly}(\lambda) \leq \text{poly}(\lambda)$;
- the constrained key size will be $|\text{ck}_C| = |\mathbf{r}_{\text{Hash}}| + |\text{ct}_k| + |\text{evk}_{\text{aHMAC}}| + n \cdot |e_s^i| \leq \text{poly}(\lambda) + \ell_k \cdot \text{poly}(\lambda) + \text{poly}(\lambda) + n \cdot \text{poly}(\log |C|, \lambda) \leq \text{poly}(\log |C|, \lambda)$.

The correctness and security of Construction 14 are established in the following theorem.

Theorem 15 (CPRF under Lattices). *The constrained PRF shown in Construction 14 is correct and 1-key selectively secure.*

Proof. We first prove the correctness. By the correctness of `batchMAC`, $\sigma_i = \mathbf{k}_i + C \cdot s_i$ over $\mathbb{Z}^{\ell_C}$. Thus, by concatenating and parsing row vectors into ring elements, $\sigma_C = \mathbf{k}_C + C \cdot s$ over $\mathcal{R}^{\ell_C}$. By the correctness of `aHMAC`, $\sigma_{C(x)} = \mathbf{k}_{C(x)} + C(x) \cdot s$ over $\mathcal{R}$. Finally by the correctness of `ExtEval`, (v_1^z, v_0^z) form shares of $C(x) \cdot \text{PRF}_x(\mathbf{k})$.

Now we prove the 1-key selective security. As a cornerstone, we need the following lemma from [44]:

Lemma 8 (Claim 5 of [44]). *Let $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}})$ be defined as in Lemma 6 and Lemma 7. For all $k \in \{0, 1\}^{\ell_k}$, the following two distributions are computationally indistinguishable:*

$$\begin{aligned} & \left(\text{pp}, \text{seed}, a, s \cdot a + e_1, s^2 \cdot a + e_2 - s \cdot \Delta, \right. \\ & \quad c_1 := r_1 \cdot a' + e_1 + k \cdot \Delta, \quad c'_1 := r_1 \cdot b' + e'_1, \\ & \quad \left. c_2 := r_2 \cdot a' + e_2, \quad c'_2 := r_2 \cdot b' + e'_2 + k \cdot \Delta. \right), \\ \text{where } & \text{seed} \xleftarrow{\$} \{0, 1\}^\lambda, a \xleftarrow{\$} \mathcal{R}_q, a' \xleftarrow{\$} \mathcal{R}_q^{\ell_k}, \\ & s, r_1, r_2 \xleftarrow{\$} \mathcal{D}_{\text{sk}}, e_1, e_2 \xleftarrow{\$} \mathcal{D}_{\text{err}}, e, e_1, e_2, e'_1, e'_2 \xleftarrow{\$} \mathcal{D}_{\text{err}}^{\ell_k}, b' := s \cdot a' + e, \end{aligned}$$

and

$$\begin{aligned} & \left(\text{pp}, \text{seed}, a, b, c, c_3, c'_3, c_4, c'_4 \right), \\ \text{where } & \text{seed} \xleftarrow{\$} \{0, 1\}^\lambda, a, b, c \xleftarrow{\$} \mathcal{R}_q, c_3, c'_3, c_4, c'_4 \xleftarrow{\$} \mathcal{R}_q^{\ell_k}. \end{aligned}$$

Now we show a series of hybrid experiments that transit from the experiment $\text{Hybrid}_0 = \text{Exp}_{\text{CPRF}}^{\mathcal{A}, 0}$ to $\text{Hybrid}_5 = \text{Exp}_{\text{CPRF}}^{\mathcal{A}, 1}$ as defined in Definition 11.

- Hybrid_0 : We summarize the view of $\mathcal{A}$ in $\text{Exp}_{\text{CPRF}}^{\mathcal{A}, 0}$ with respect to a challenge constraint C .
 1. As public parameters, $\mathcal{A}$ receives the lattice public parameters $\text{pp}_{\text{HSS}} = (\mathcal{R}, p, q, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}})$, the batch-MAC public key $\text{pk}_{\text{batchMAC}}$, and a PRG seed seed ;
 2. As the constrained key, $\mathcal{A}$ receives a random r_{Hash} used in batchMAC.Hash , a ciphertext ct_k as in Lemma 7, an aHMAC evaluation key $\text{evk}_{\text{aHMAC}}$ as in Lemma 6, and compressive MACs e_s ;
 3. For any number of queries $x \in \{0, 1\}^\lambda$, $\mathcal{A}$ receives evaluations $v_0^z \leftarrow \text{Eval}(\text{pp}, \text{msk}, x)$.
- Hybrid_1 : In this hybrid, instead of honestly computing the evaluation of x as $v_0^z \leftarrow \text{Eval}(\text{pp}, \text{msk}, x)$, simulate it by computing

$$\begin{aligned} v_1^z & \leftarrow \text{CEval}(\text{pp}, \text{ck}_C, C, x), \\ v_0^z & \leftarrow v_1^z - C(x) \cdot \text{PRF}(k, x). \end{aligned}$$

$\text{Hybrid}_1 \stackrel{c}{\approx} \text{Hybrid}_0$ because of the correctness of the constrained PRF. Note that in this hybrid, the evaluations do not use the aHMAC secret key sk_{aHMAC} . Thus the key s is only used in ct_k , $\text{evk}_{\text{aHMAC}}$ and e_s .

- Hybrid_2 : In this hybrid, instead of computing e_s by running batchMAC.Auth on s , simulate it by invoking the simulator of the message privacy of batchMAC . Namely,

$$e_s \xleftarrow{\$} \text{Sim}_0(1^\lambda, \text{pk}_{\text{batchMAC}}, C, r_{\text{Hash}}).$$

$\text{Hybrid}_2 \stackrel{c}{\approx} \text{Hybrid}_1$ by the message privacy of the underlying batchMAC . Note that in this hybrid, the global secret key s is only used in ct_k and $\text{evk}_{\text{aHMAC}}$.

- **Hybrid₃**: In this hybrid, instead of computing ct_k and $\text{evk}_{\text{aHMAC}}$ honestly, simulate them by simply sampling random values (except pp_{HSS}).
 $\text{Hybrid}_3 \stackrel{c}{\approx} \text{Hybrid}_2$ by combining Lemma 6, Lemma 7, and Lemma 8. Note that in this hybrid, the PRF key k is only used for evaluating $\text{PRF}(k, x)$, and nowhere else.
- **Hybrid₄**: In this hybrid, instead of answering the challenge query x^* , satisfying $C(x^*) = 1$, by computing $(v_0^z)^* \leftarrow (v_1^z)^* - \text{PRF}(k, x^*)$, simulate it as $(v_0^z)^* \xleftarrow{\$} \mathcal{R}$.
 $\text{Hybrid}_4 \stackrel{c}{\approx} \text{Hybrid}_3$ by the security of PRF.
- **Hybrid₅**: This is the experiment $\text{Exp}_{\text{CPRF}}^{\text{A},1}$. The same arguments from Hybrid₁ to Hybrid₄, in reverse order, shows that $\text{Hybrid}_5 \stackrel{c}{\approx} \text{Hybrid}_4$.

By a hybrid argument, we conclude that $\text{Hybrid}_0 \stackrel{c}{\approx} \text{Hybrid}_5$, which proves the 1-key selective security.

CPRF under Paillier Groups. Alternatively, by combining our DCR-based batch-MAC (Construction 7) with Paillier-group-based aHMAC and HSS, we obtain a CPRF under Paillier groups. We first formalize our result as the following theorem.

Theorem 16 (CPRF under Paillier Groups). *Assuming the circular power-DDH assumption in Paillier groups, there exists a 1-key selectively secure constrained PRF. Its master secret key size is $O(|C|^{2/3}\lambda) + \text{poly}(\lambda)$. Its constrained key size is also $O(|C|^{2/3}\lambda) + \text{poly}(\lambda)$.*

This construction is obtained by incorporating our DCR-based batch-MAC scheme (Construction 7) with the aHMACs and extended HSS schemes built in Paillier groups.

Recall that in lattice-based instantiations of aHMAC and HSS, the secret key s is a polynomial in $\mathcal{R} := \mathbb{Z}[X]/(X^n + 1)$. Thus, we have to invoke batch-MAC n times, each using a single coefficient of s as the MAC key. However, in Paillier-based instantiations of aHMAC and HSS, the secret key s is a single integer in $[N]$, so we only need one batch-MAC invocation.

We summarize the aHMACs and extended HSS evaluations under Paillier groups in the following lemmas.

Lemma 9 (Algebraic Homomorphic MACs under Paillier Groups [43]).

Let $B < 2^{\text{poly}(\lambda)}$ be a bound on input values. Let $\text{pp} := (G, F, H, f, N, \ell)$ be the public parameters of a Paillier group (Lemma 1) with $N \gg B$. Let $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ be a PRG seed.

Assuming the Circular Power-DDH (CP-DDH) assumption in Paillier groups, there exists an algebraic homomorphic MAC scheme aHMAC for arbitrary polynomial-sized circuit, within which

- *the global secret key is $s \xleftarrow{\$} [N]$,*

- for input $x \leq B$, the MAC key is $k_x \in [N]$, the MAC tag is $\sigma_x := k_x + s \cdot x$ computed in $\mathbb{Z}$

Further, aHMAC consists of three efficient algorithms:

- $\text{KGen}(1^\lambda, s)$ takes as input the global secret key s , and outputs a secret key sk and an evaluation key evk .
- $\text{EvalTag}(\text{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$ takes as inputs the evaluation key evk , a circuit $C : \mathbb{Z}_B^{\ell_x} \rightarrow \mathbb{Z}_B^{\ell_z}$, input values $\mathbf{x} = (x_1, \dots, x_{\ell_x}) \in \mathbb{Z}_B^{\ell_x}$, and their associated tags $\sigma_{\mathbf{x}} = (\sigma_{x_1}, \dots, \sigma_{x_{\ell_x}})$. It outputs tags $\sigma_{\mathbf{z}} = (\sigma_{z_1}, \dots, \sigma_{z_{\ell_z}}) \in \mathbb{Z}^{\ell_z}$ authenticating $\mathbf{z} := C(\mathbf{x})$.
- $\text{EvalKey}(\text{sk}, C, k_{\mathbf{x}})$ takes as inputs the secret key sk , a circuit $C : \mathbb{Z}_B^{\ell_x} \rightarrow \mathbb{Z}_B^{\ell_z}$ and the input keys $k_{\mathbf{x}} = (k_{x_1}, \dots, k_{x_{\ell_x}})$. It outputs MAC keys $k_{\mathbf{z}} = (k_{z_1}, \dots, k_{z_{\ell_z}}) \in \mathbb{Z}^{\ell_z}$ for $\mathbf{z} := C(\mathbf{x})$.

Specifically, the evaluation key evk is formulated as

$$\begin{aligned} \text{pp} &= (G, F, H, f, N, \ell), (g, \rho) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}), \text{seed} \xleftarrow{\$} \{0, 1\}^\lambda, \\ s &\xleftarrow{\$} [N], \mathbf{r} \xleftarrow{\$} [N]^{\lceil \log N \rceil}, \\ \text{evk} &:= (\text{pp}, \text{seed}, g^{\mathbf{r}}, g^{\mathbf{r} \cdot s}, g^{\mathbf{r} \cdot s^2} \odot f^{\text{Bin}(s)}). \end{aligned}$$

The aHMAC scheme satisfies correctness, namely for all $\lambda \in \mathbb{N}$, and any input $\mathbf{x}$:

$$\Pr \left[\begin{array}{c} \sigma_{\mathbf{x}} \neq k_{\mathbf{x}} + s \cdot \mathbf{x} \\ \text{(over } \mathbb{Z}) \end{array} \middle| \begin{array}{l} \sigma_{\mathbf{x}} = k_{\mathbf{x}} + s \cdot \mathbf{x} \\ (\text{evk}, \text{sk}) \xleftarrow{\$} \text{KGen}(1^\lambda, s) \\ \mathbf{z} = C(\mathbf{x}) \\ \sigma_{\mathbf{z}} \leftarrow \text{EvalTag}(\text{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}) \\ k_{\mathbf{z}} \leftarrow \text{EvalKey}(\text{sk}, C, k_{\mathbf{x}}) \end{array} \right] \leq \text{negl}(\lambda)$$

Lemma 10 (Extended HSS Evaluation under Paillier Groups [1, 31, 57, 63]). Set B and $\text{pp} = (G, F, H, f, N, \ell)$ as in Lemma 9. Let $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ be a PRG seed. Assuming the DCR assumption, there exists a pair of efficient deterministic algorithms, ExtEval_0 and ExtEval_1 , such that: for every logarithmic function $d(\lambda)$, every polynomial $p(\lambda)$, there exists a negligible function $\text{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, Boolean circuit $C : \{0, 1\}^{\ell_y} \rightarrow \{0, 1\}$ with $|C| < p(\lambda)$, $\text{Depth}(C) < d(\lambda)$ ⁵, global secret key $s \in [N]$, inputs $x \in \mathbb{Z}_B$ and $\mathbf{y} \in \{0, 1\}^{\ell_y}$, and additive shares (over $\mathbb{Z}$) $\langle s \cdot x \rangle_1 - \langle s \cdot x \rangle_0 = sx$, $\langle x \rangle_1 - \langle x \rangle_0 = x$, the following holds:

$$\Pr \left[\begin{array}{c} w_1^z = w_0^z + s \cdot z \\ v_1^z = v_0^z + z \\ \text{(over } \mathbb{Z}) \end{array} \middle| \begin{array}{l} (w_b^z, v_b^z) \leftarrow \text{ExtEval}_b(\text{pp}, \text{seed}, \text{ct}_{\mathbf{y}}, C, \langle s \cdot x \rangle_b, \langle x \rangle_b) \\ z := x \cdot C(\mathbf{y}) \text{ (over } \mathbb{Z}) \end{array} \right] > 1 - \text{negl}(\lambda),$$

⁵ We use $\text{Depth}(C)$ to denote the depth of circuit C .

over the randomness of seed and ct_y . Further, the ciphertext ct_y is computed as follows:

$$(g, \rho) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}), \mathbf{r}', \mathbf{r}'' \xleftarrow{\$} [N]^{\ell_y},$$

$$\text{ct}_y := (g^{\mathbf{r}'}, g^{\mathbf{r}' \cdot s} \odot f^y, g^{\mathbf{r}'' \cdot s}, g^{\mathbf{r}''} \odot f^y).$$

Again, our CPRF works for the class of polynomial-sized circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ where

$$\mathcal{C}_\lambda := \{C : \{0, 1\}^\lambda \rightarrow \{0, 1\} : |C| < \text{poly}(\lambda)\}.$$

We use ℓ_C to denote the length of C when described as a boolean vector. As shorthands, in the following we write $U_x(\cdot) = U(\cdot, x)$, and $\text{PRF}_x(\cdot) = \text{PRF}(\cdot, x)$, where U is a universal circuit such that $U_x(C) = C(x)$ for all circuits $C \in \mathcal{C}_\lambda$. Specifically, the domain of the CPRF is $\mathcal{X} := \{0, 1\}^\lambda$ and the range is $\mathbb{Z}$.

Construction 17 (Constrained PRF under Paillier Groups) *Given the following ingredients:*

- a batch-MAC scheme $\text{batchMAC} = (\text{Setup}, \text{KeyGen}, \text{Hash}, \text{Auth}, \text{Ext})$, as defined in Definition 3, for message space $\mathbb{Z}_N^{\ell_C}$, where N is the easy-group-order of the Paillier instantiation for aHMAC and extended HSS;
- an aHMAC scheme $\text{aHMAC} = (\text{KGen}, \text{EvalTag}, \text{EvalKey})$ as shown in Lemma 9;
- a pair of extended HSS evaluation algorithms $(\text{ExtEval}_0, \text{ExtEval}_1)$ as shown in Lemma 10;
- a secure PRF $\text{PRF} : \mathcal{K} \times \{0, 1\}^\lambda \rightarrow \mathbb{Z}$ with evaluation algorithm in NC1, and key space $\mathcal{K} = \{0, 1\}^{\ell_k}$,

we construct a constrained PRF $\text{CPRF} = (\text{KeyGen}, \text{Eval}, \text{Constrain}, \text{CEval})$ as follows:

- $\text{KeyGen}(1^\lambda)$:
 1. Set the public parameter $\text{pp}_{\text{HSS}} := (G, F, H, f, N, \ell)$ as in Lemma 9 and Lemma 10.
 2. Sample a PRG seed $\text{seed} \xleftarrow{\$} \{0, 1\}^\lambda$ for aHMAC and HSS evaluation.
 3. Compute $\text{pk}_{\text{batchMAC}} \xleftarrow{\$} \text{batchMAC.Setup}(1^\lambda)$.
 4. Sample a PRF key for PRF, $\mathbf{k} \xleftarrow{\$} \mathcal{K} = \{0, 1\}^{\ell_k}$.
 5. Compute the HSS ciphertext for $\mathbf{k}$ as in Lemma 10, and denote it as $\text{ct}_\mathbf{k}$.
 6. Sample an HSS global secret key $s \xleftarrow{\$} [N]$.
 7. Compute $(\text{sk}_{\text{aHMAC}}, \text{evk}_{\text{aHMAC}}) \xleftarrow{\$} \text{aHMAC.KGen}(1^\lambda, s)$.
 8. Compute the MAC key for C :

$$(\mathbf{k}, \text{st}_A) \xleftarrow{\$} \text{batchMAC.KeyGen}(\text{pk}_{\text{batchMAC}}).$$

9. Collect the randomness in batchMAC.KeyGen algorithm and denote it as a vector $\mathbf{r}_{\text{MAC}}$.
10. Return

$$\text{pp} := (\text{pp}_{\text{HSS}}, \text{seed}, \text{pk}_{\text{batchMAC}}),$$

and

$$\text{msk} := (\text{ct}_\mathbf{k}, \text{sk}_{\text{aHMAC}}, \text{evk}_{\text{aHMAC}}, \mathbf{r}_{\text{MAC}}).$$

– Eval(pp, msk, $x \in \{0, 1\}^\lambda$):

1. *Recompute*

$$(\mathbf{k}, \text{st}_A) \xleftarrow{\$} \text{batchMAC.KeyGen}(\text{pk}_{\text{batchMAC}}),$$

using randomness $\mathbf{r}_{\text{MAC}}$.

Denote $\mathbf{k}$ as $k_C \in \mathbb{Z}^{\ell_C}$.

2. *Compute*

$$k_{C(x)} \leftarrow \text{aHMAC.EvalKey}(\text{sk}_{\text{aHMAC}}, U_x, k_C).$$

3. *Compute*

$$(w_0^z, v_0^z) \leftarrow \text{ExtEval}_0(\text{pp}_{\text{HSS}}, \text{seed}, \text{ct}_k, \text{PRF}_x, k_{C(x)}, 0).$$

4. *Return* v_0^z .

– Constrain(pp, msk, C):

1. *Recompute*

$$(\mathbf{k}, \text{st}_A) \xleftarrow{\$} \text{batchMAC.KeyGen}(\text{pk}_{\text{batchMAC}}),$$

using randomness $\mathbf{r}_{\text{MAC}}$.

2. *Compute*

$$(d_C, \text{st}_U) \xleftarrow{\$} \text{batchMAC.Hash}(\text{pk}_{\text{batchMAC}}, C)$$

and denote the randomness used in the algorithm as $\mathbf{r}_{\text{Hash}}$.

3. *Compute compressive MAC for key s :*

$$e_s \xleftarrow{\$} \text{batchMAC.Auth}(\text{pk}_{\text{batchMAC}}, s, d_C, \text{st}_A)$$

4. *Return*

$$\text{ck}_C := (\mathbf{r}_{\text{Hash}}, \text{ct}_k, \text{evk}_{\text{aHMAC}}, e_s).$$

– CEval(pp, ck_C , C , x):

1. *Recompute*

$$(d_C, \text{st}_U) \xleftarrow{\$} \text{batchMAC.Hash}(\text{pk}_{\text{batchMAC}}, C)$$

using randomness $\mathbf{r}_{\text{Hash}}$.

2. *Compute the MAC tags for C :*

$$\sigma \leftarrow \text{batchMAC.Ext}(\text{pk}_{\text{batchMAC}}, e_s, \text{st}_U).$$

Denote σ as $\sigma_C \in \mathbb{Z}^{\ell_C}$.

3. *Compute*

$$\sigma_{C(x)} \leftarrow \text{aHMAC.EvalTag}(\text{evk}_{\text{aHMAC}}, U_x, C, \sigma_C).$$

4. *Compute*

$$(w_1^z, v_1^z) \leftarrow \text{ExtEval}_1(\text{pp}_{\text{HSS}}, \text{seed}, \text{ct}_k, \text{PRF}_x, \sigma_{C(x)}, C(x)).$$

5. *Return* v_1^z .

Analysis of key sizes. If we instantiate the underlying batchMAC under the DCR assumption (i.e. Construction 7),

- the master secret key size $|\text{msk}| = |\text{ct}_k| + |\text{sk}_{\text{aHMAC}}| + |\text{evk}_{\text{aHMAC}}| + |\text{r}_{\text{MAC}}| \leq \ell_k \cdot \text{poly}(\lambda) + \text{poly}(\lambda) + \text{poly}(\lambda) + O(|C|^{\frac{2}{3}} \cdot \lambda) \leq O(|C|^{\frac{2}{3}}) + \text{poly}(\lambda)$;
- the constrained key size $|\text{ck}_C| = |\text{r}_{\text{Hash}}| + |\text{ct}_k| + |\text{evk}_{\text{aHMAC}}| + |e_s| \leq O(|C|^{\frac{2}{3}} \cdot \lambda) + \ell_k \cdot \text{poly}(\lambda) + \text{poly}(\lambda) + O(|C|^{\frac{2}{3}} \cdot \lambda) \leq O(|C|^{\frac{2}{3}}) + \text{poly}(\lambda)$.

The security proof is almost identical. The only difference is that we need the following lemma as a counterpart of Lemma 8.

Lemma 11 (Claim 1 of [44]). *Let $\text{pp} = (G, F, H, f, N, \ell)$ be defined as in Lemma 9 and Lemma 10. Assuming the circular power-DDH assumption holds in Paillier groups, then for all $\mathbf{k} \in \{0, 1\}^{\ell_k}$, the following two distributions are computationally indistinguishable:*

$$\begin{aligned} & \left(\text{pp}, \text{seed}, g^{\mathbf{r}}, g^{\mathbf{r} \cdot s}, g^{\mathbf{r} \cdot s^2} \odot f^{\text{Bin}(s)} \right. \\ & \quad \left. g^{\mathbf{r}'}, g^{\mathbf{r}' \cdot s} \odot f^{\mathbf{k}}, g^{\mathbf{r}'' \cdot s}, g^{\mathbf{r}''} \odot f^{\mathbf{k}} \right) \\ \text{where } & (g, \rho) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}), s \xleftarrow{\$} [N], \text{seed} \xleftarrow{\$} \{0, 1\}^\lambda, \\ & \mathbf{r} \xleftarrow{\$} [N]^{\lceil \log N \rceil}, \mathbf{r}', \mathbf{r}'' \xleftarrow{\$} [N]^{\ell_k}, \end{aligned}$$

and

$$\begin{aligned} & \left(\text{pp}, \text{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}, g^{\mathbf{a}'}, g^{\mathbf{b}'}, g^{\mathbf{a}''}, g^{\mathbf{b}''} \right) \\ \text{where } & (g, \rho) \xleftarrow{\$} \mathcal{D}(1^\lambda, \text{pp}), \text{seed} \xleftarrow{\$} \{0, 1\}^\lambda, \\ & \mathbf{a}, \mathbf{b}, \mathbf{c} \xleftarrow{\$} [\ell]^{\lceil \log N \rceil}, \mathbf{a}', \mathbf{b}', \mathbf{a}'', \mathbf{b}'' \xleftarrow{\$} [\ell]^{\ell_k}. \end{aligned}$$

D Succinct 2PC Protocol

A recent line of work has shown that by leveraging the homomorphic property of MACs with the form $\sigma = \mathbf{x} \cdot \Delta + \mathbf{k}$, one can build highly succinct garbled circuits, also called *1-bit-per-gate garbling* schemes [44, 51, 54].

By applying our batch-MAC scheme to generate the MACs in 1-bit-per-gate garbling schemes, we acquire a 2-round secure 2-party computation protocol. To be specific, suppose Alice holds input $\mathbf{x}$ and Bob holds input $\mathbf{y}$, where $|\mathbf{x}| = m$ and $|\mathbf{y}| = \ell$, and $C(\cdot, \cdot)$ is an arbitrary public circuit. During the protocol, Alice first sends a message to Bob, then Bob sends back a message to Alice. After the interaction, Alice learns the output $C(\mathbf{x}, \mathbf{y})$, while keeping both parties' inputs private. Our protocol further enjoys the following merits:

- Alice's message is sublinear in m ;
- Bob's message consists of three parts. The first part is sublinear in m . The second part is rate-1 to ℓ , and the third part is rate-1 to $|C|$.
- Moreover, the third part of Bob's message is independent of inputs $\mathbf{x}$ and $\mathbf{y}$, thus can be computed and published during a preprocessing phase. In this case, the online computation of Bob is independent of $|C|$;

- Alice’s privacy holds statistically.

We begin by formally defining a 2-round secure 2-party computation protocol.

Definition 12 (2-round 2PC). Let λ be the security parameter, $m(\lambda)$ be Alice’s input length, and $\ell(\lambda)$ be Bob’s input length. Let $\mathbf{x} \in \{0, 1\}^m$ be Alice’s input, and $\mathbf{y} \in \{0, 1\}^\ell$ be Bob’s input.

Let $\mathcal{C} = \{C : \{0, 1\}^{m+\ell} \rightarrow \{0, 1\} : |C| < \text{poly}(\lambda)\}$ be the class of polynomial-sized boolean circuits. A 2-round secure 2-party computation (2-round 2PC) protocol for $\mathcal{C}$ consists of the following four efficient algorithms:

- $\text{Setup}(1^\lambda) \rightarrow \text{crs}$: The setup algorithm is probabilistic. It takes a security parameter λ as input and outputs a common reference string crs .
- $\text{Query}(\text{crs}, \mathbf{x}) \rightarrow (\text{query}, \text{st})$: The query algorithm is probabilistic. It takes as input the crs and Alice’s input $\mathbf{x}$, outputs a query message query and a private state st .
- $\text{Ans}(\text{crs}, \text{query}, \mathbf{y}, C \in \mathcal{C}) \rightarrow \text{ans}$: The answer algorithm is probabilistic. It takes as input the crs , query , Bob’s input $\mathbf{y}$ and the circuit C , outputs an answer message ans .
- $\text{Eval}(\text{crs}, \text{st}, \text{ans}, C) \rightarrow z$: The evaluation algorithm is deterministic. It takes as input the crs , st , ans , and C , outputs a boolean value z .

Further, it should satisfy the following properties:

Correctness. For all security parameter $\lambda \in \mathbb{N}$, every pair of inputs $\mathbf{x}$ and $\mathbf{y}$, and all $C \in \mathcal{C}$, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\Pr \left[\begin{array}{l} \text{crs} \xleftarrow{\$} \text{Setup}(1^\lambda) \\ (\text{query}, \text{st}) \xleftarrow{\$} \text{Query}(\text{crs}, \mathbf{x}) \\ \text{ans} \xleftarrow{\$} \text{Ans}(\text{crs}, \text{query}, \mathbf{y}, C) \\ z \leftarrow \text{Eval}(\text{crs}, \text{st}, \text{ans}, C) \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

Statistical Alice’s Privacy. For all security parameter $\lambda \in \mathbb{N}$, and any $\mathbf{x}_0, \mathbf{x}_1 \in \{0, 1\}^m$, the following two distributions are statistically close:

$$\left\{ \text{query} \middle| \begin{array}{l} \text{crs} \xleftarrow{\$} \text{Setup}(1^\lambda) \\ (\text{query}, \text{st}) \xleftarrow{\$} \text{Query}(\text{crs}, \mathbf{x}_0) \end{array} \right\}, \left\{ \text{query} \middle| \begin{array}{l} \text{crs} \xleftarrow{\$} \text{Setup}(1^\lambda) \\ (\text{query}, \text{st}) \xleftarrow{\$} \text{Query}(\text{crs}, \mathbf{x}_1) \end{array} \right\}.$$

Bob’s Privacy. Consider the following experiment $\text{Exp}_{2\text{PC}}^{\mathcal{A}, \text{Sim}}(1^\lambda)$ parametrized by an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ and a simulator Sim :

1. Sample a common reference string $\text{crs} \xleftarrow{\$} \text{Setup}(1^\lambda)$.
2. Activate the adversary $(\mathbf{x}, \mathbf{y}, \text{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda, \text{crs})$.
3. Compute $(\text{query}, \text{st}) \xleftarrow{\$} \text{Query}(\text{crs}, \mathbf{x})$ and let $\mathbf{r}_0$ denote the randomness consumed.
4. Sample a random bit $b \xleftarrow{\$} \{0, 1\}$.
5. If $b = 0$ compute $\text{ans}_0 \xleftarrow{\$} \text{Ans}(\text{crs}, \text{query}, \mathbf{y}, C)$,
else compute $\text{ans}_1 \xleftarrow{\$} \text{Sim}(1^\lambda, \text{crs}, \mathbf{x}, \mathbf{r}_0, C, C(\mathbf{x}, \mathbf{y}))$.

6. Compute $b' \leftarrow \mathcal{A}_1(\text{ans}_b, \mathbf{r}_0, \text{aux})$.
7. Return 1 if and only if $b' = b$.

We say that a 2-round 2PC (Setup, Query, Ans, Eval) satisfies Bob's privacy if there exists a PPT simulator Sim , such that for any PPT adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, we have that

$$\left| \Pr \left[\text{Exp}_{2\text{PC}}^{\mathcal{A}, \text{Sim}} = 1 \right] - \frac{1}{2} \right| \leq \text{negl}(\lambda).$$

Specifically, we apply the 1-bit-per-gate garbling scheme proposed by Ishai, Li, and Lin [44]. In their work, they showed constructions under different style assumptions, including prime-order groups, DCR, and lattices. We summarize their scheme under the lattice assumption in Lemma 12.

Within a circuit C , each wire corresponds to an index. We use $\text{Wire}(x)$ to denote the wire (or rather, the index) that bears the value x , and $\text{Wire}(\mathbf{x})$ to denote a vector consisting of $\text{Wire}(\mathbf{x}[i])$ for each $i \in [|\mathbf{x}|]$. $\text{Wire}(C)$ denotes the set of all the wires in C . For a gate g in a circuit, we use $\text{InWires}(g)$ to denote the input wires of g , and use $\text{OutWire}(g)$ to denote the output wire of g (we only consider gates with a single output). We sometimes abuse these notations as $\text{InWires}(C)$ and $\text{OutWire}(C)$ to denote the input and output wires of C .

Lemma 12 (1-bit-per-gate Garbling under Lattices [44]). *Set the lattice public parameters $\text{pp} := (\mathcal{R}, p, q, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}})$ as in Lemma 6, namely $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, $n(\lambda) \leq \text{poly}(\lambda)$ is a power-of-2, $p \geq \lambda^{\omega(1)}$, $q = p \cdot \Delta$, $\Delta \geq p \cdot \lambda^{\omega(1)}$, $\mathcal{D}_{\text{sk}}$ and $\mathcal{D}_{\text{err}} \subseteq \mathcal{R}$ have coefficients bounded by $\text{poly}(\lambda)$.*

Further, set the garbling keys and labels as follows:

- Sample a global key $s \xleftarrow{\$} \mathcal{D}_{\text{sk}}$, which is used to generate garbling labels;
- Let $\text{PRF} : \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \{0, 1\}$ be a secure PRF in NC1. Sample a PRF key $\text{sk} \xleftarrow{\$} \{0, 1\}^\lambda$;
- The inputs consist of two parts. For the first part $\mathbf{x} \in \{0, 1\}^m$, the evaluator knows $\mathbf{x}$. For the second part $\mathbf{y} \in \{0, 1\}^\ell$, the evaluator acquires its one-time-pad $\bar{\mathbf{y}} := \mathbf{y} \oplus \text{PRF}(\text{sk}, \text{Wire}(\mathbf{y}))$;
- Sample the garbling keys for each input. $K_{\mathbf{x}} \xleftarrow{\$} \mathcal{R}_p^m$, $K_{\bar{\mathbf{y}}} \xleftarrow{\$} \mathcal{R}_p^\ell$;
- Set the garbling labels as $L_{\mathbf{x}} = K_{\mathbf{x}} + s \cdot \mathbf{x}$, $L_{\bar{\mathbf{y}}} = K_{\bar{\mathbf{y}}} + s \cdot \bar{\mathbf{y}}$, computed in $\mathcal{R}$.

Given the above conditions, and assuming the circular power ring-LWE (CP-RLWE) assumption, there exist three efficient algorithms:

- $\text{Gen}(\text{pp}, s, \text{sk}) \rightarrow \text{pd}$: The generation algorithm is probabilistic. It takes as input the public parameter pp , global key s , the PRF key sk , and outputs the public data pd .
- $\text{Gb}(\text{pd}, s, \text{sk}, K_{\mathbf{x}}, K_{\bar{\mathbf{y}}}, C) \rightarrow \hat{C}$: The garbling algorithm is deterministic. It takes as input the pd , s , sk , input keys $K_{\mathbf{x}}$ and $K_{\bar{\mathbf{y}}}$, the circuit C , output the garbled circuit $\hat{C}$. Note that $\hat{C}$ is input-independent.

- $\text{Ev}(\text{pd}, L_{\mathbf{x}}, \mathbf{x}, L_{\bar{\mathbf{y}}}, \bar{\mathbf{y}}, C, \hat{C}) \rightarrow z$: The evaluation algorithm is deterministic. It takes as input the pd , the input values $\mathbf{x}$ and corresponding labels $L_{\mathbf{x}}$, the masked input values $\bar{\mathbf{y}}$ and corresponding labels $L_{\bar{\mathbf{y}}}$, the circuit C and the garbled circuit $\hat{C}$. It outputs a bit $z \in \{0, 1\}$.

Moreover, the size of pd is $\text{poly}(\lambda)$, and the size of $\hat{C}$ is exactly $|C|$. We say the correctness holds if $z = C(\mathbf{x}, \mathbf{y})$ with overwhelming probability, with randomness taken from sampling s, sk and the Gen algorithm.

Our incorporation of batch-MAC and 1-bit-per-gate garbling is straightforward. We view each coefficient of the global garbling key $s \in \mathcal{R}$ as a global MAC key s_i , for $i = 1, \dots, n$. We apply our batch-MAC scheme to transfer the MACs $\sigma_0^i = \mathbf{x} \cdot s_i + \mathbf{k}_0^i$ and $\sigma_1^i = \bar{\mathbf{y}} \cdot s_i + \mathbf{k}_1^i$ from the garbler to the evaluator. They can then concatenate them to acquire $L_{\mathbf{x}} = \mathbf{x} \cdot s + K_{\mathbf{x}}$ and $L_{\bar{\mathbf{y}}} = \bar{\mathbf{y}} \cdot s + K_{\bar{\mathbf{y}}}$ as the garbled input labels.

Construction 18 (Succinct 2-Round 2PC) *Given the following ingredients:*

- two batch-MAC schemes $\text{batchMAC} = (\text{Setup}, \text{KeyGen}, \text{Hash}, \text{Auth}, \text{Ext})$ as defined in Definition 3 for message space $\mathbb{Z}_B^m$ and $\mathbb{Z}_B^\ell$ respectively, where B is an upper bound for $\mathcal{D}_{\text{sk}}$'s coefficients;
- a 1-bit-per-gate garbling scheme $\text{bitGC} = (\text{Gen}, \text{Gb}, \text{Ev})$ for arbitrary boolean circuit C , as described in Lemma 12,

we construct a 2-round 2PC protocol $(\text{Setup}, \text{Query}, \text{Ans}, \text{Eval})$ as follows:

- $\text{Setup}(1^\lambda)$:
 1. Sample public parameters $\text{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\text{sk}}, \mathcal{D}_{\text{err}})$ as in Lemma 12.
 2. Compute $\text{pk} \xleftarrow{\$} \text{batchMAC.Setup}(1^\lambda)$.
 3. Return $\text{crs} := (\text{pp}, \text{pk})$.
- $\text{Query}(\text{crs}, \mathbf{x})$:
 1. Compute

$$(d_{\mathbf{x}}, \text{st}_U^0) \xleftarrow{\$} \text{batchMAC.Hash}(\text{pk}, \mathbf{x}).$$
 2. Return $(\text{query}, \text{st}) := (d_{\mathbf{x}}, (\text{st}_U^0, \mathbf{x}))$.
- $\text{Ans}(\text{crs}, \text{query}, \mathbf{y}, C)$:
 1. Sample a global garbling key $s \xleftarrow{\$} \mathcal{D}_{\text{sk}}$.
 2. Sample a PRF key $\text{sk} \xleftarrow{\$} \{0, 1\}^\lambda$ for an NC1 PRF PRF.
 3. Compute $\text{pd} \xleftarrow{\$} \text{bitGC.Gen}(\text{pp}, s, \text{sk})$.
 4. Parse $s \in \mathcal{R}$ naturally as an n -dimension vector $\mathbf{s} = (s_1, \dots, s_n) \in \mathbb{Z}^n$. For $i = 1, \dots, n$, compute MAC keys for $\mathbf{x}$:

$$(\mathbf{k}_0^i, \text{st}_{A,0}^i) \xleftarrow{\$} \text{batchMAC.KeyGen}(\text{pk}).$$

Concatenate $\mathbf{k}_0^i$ for $i = 1, \dots, n$ as an $m \times n$ matrix.

Parse the j -th row of this matrix as a vector $\mathbf{v}_0^j \in \mathbb{Z}^n$ for $j = 1, \dots, m$.

Parse each vector $\mathbf{v}_0^j$ naturally as a ring element in $\mathcal{R}$, and denote them altogether as $K_{\mathbf{x}} \in \mathcal{R}^m$.

5. Compute

$$e_{\mathbf{x}}^i \xleftarrow{\$} \text{batchMAC.Auth}(\text{pk}, s_i, d_{\mathbf{x}}, \text{st}_{A,0}^i),$$

for $i = 1, \dots, n$.

6. Compute $\bar{\mathbf{y}} := \mathbf{y} \oplus \text{PRF}(\text{sk}, \text{Wire}(\mathbf{y}))$.

7. For $i = 1, \dots, n$, compute MAC keys for $\bar{\mathbf{y}}$:

$$(\mathbf{k}_1^i, \text{st}_{A,1}^i) \xleftarrow{\$} \text{batchMAC.KeyGen}(\text{pk}).$$

Concatenate $\mathbf{k}_1^i$ for $i = 1, \dots, n$ as an $\ell \times n$ matrix.

Parse the j -th row of this matrix as a vector $\mathbf{v}_1^j \in \mathbb{Z}^n$ for $j = 1, \dots, \ell$.

Parse each vector $\mathbf{v}_1^j$ naturally as a ring element in $\mathcal{R}$, and denote them altogether as $K_{\bar{\mathbf{x}}} \in \mathcal{R}^\ell$.

8. Compute

$$(d_{\bar{\mathbf{y}}}, \text{st}_U^1) \xleftarrow{\$} \text{batchMAC.Hash}(\text{pk}, \bar{\mathbf{y}}),$$

and denote the randomness consumed as $\mathbf{r}$.

9. Compute

$$e_{\bar{\mathbf{y}}}^i \xleftarrow{\$} \text{batchMAC.Auth}(\text{pk}, s_i, d_{\bar{\mathbf{y}}}, \text{st}_{A,1}^i),$$

for $i = 1, \dots, n$.

10. Compute

$$\hat{C} \leftarrow \text{bitGC.Gb}(\text{pd}, s, \text{sk}, K_{\mathbf{x}}, K_{\bar{\mathbf{y}}}, C).$$

11. Return $\text{ans} := \left((e_{\mathbf{x}}^i)_{i=1}^n, \bar{\mathbf{y}}, \mathbf{r}, (e_{\bar{\mathbf{y}}}^i)_{i=1}^n, \text{pd}, \hat{C} \right)$.

– Eval($\text{crs}, \text{st}, \text{ans}, C$):

1. For $i = 1, \dots, n$, compute MAC tags for $\mathbf{x}$:

$$\sigma_0^i \leftarrow \text{batchMAC.Ext}(\text{pk}, e_{\mathbf{x}}^i, \text{st}_U^0).$$

Concatenate σ_0^i for $i = 1, \dots, n$ as an $m \times n$ matrix.

Parse the j -th row of this matrix as a vector $\mathbf{w}_0^j \in \mathbb{Z}^n$ for $j = 1, \dots, m$.

Parse each vector $\mathbf{w}_0^j$ naturally as a ring element in $\mathcal{R}$, and denote them altogether as $L_{\mathbf{x}} \in \mathcal{R}^m$.

2. Recompute

$$(d_{\bar{\mathbf{y}}}, \text{st}_U^1) \leftarrow \text{batchMAC.Hash}(\text{pk}, \bar{\mathbf{y}})$$

using randomness $\mathbf{r}$.

For $i = 1, \dots, n$, compute MAC tags for $\bar{\mathbf{y}}$:

$$\sigma_1^i \leftarrow \text{batchMAC.Ext}(\text{pk}, e_{\bar{\mathbf{y}}}^i, \text{st}_U^1).$$

Concatenate σ_1^i for $i = 1, \dots, n$ as an $\ell \times n$ matrix.

Parse the j -th row of this matrix as a vector $\mathbf{w}_1^j \in \mathbb{Z}^n$ for $j = 1, \dots, \ell$.

Parse each vector $\mathbf{w}_1^j$ naturally as a ring element in $\mathcal{R}$, and denote them altogether as $L_{\bar{\mathbf{y}}} \in \mathcal{R}^\ell$.

3. Compute

$$z \leftarrow \text{bitGC.Ev}(\text{pd}, L_{\mathbf{x}}, \mathbf{x}, L_{\bar{\mathbf{y}}}, \bar{\mathbf{y}}, C, \hat{C}).$$

4. Return z .

Analysis of efficiency. When we instantiate the underlying batch-MAC with our LWE-based construction (Construction 9), the query size $|\text{query}| = |d_{\mathbf{x}}| \leq \text{poly}(\log m, \lambda)$. The answer ans consists of two parts. The first part, which is independent of the circuit C , contains $(e_{\mathbf{x}}^i)_{i=1}^n, \bar{\mathbf{y}}, \mathbf{r}, (e_{\bar{\mathbf{y}}}^i)_{i=1}^n$, and is of size $n \cdot \text{poly}(\log m, \lambda) + \ell + \text{poly}(\lambda) + n \cdot \text{poly}(\log \ell, \lambda) \leq \text{poly}(\log m, \lambda) + \text{poly}(\log \ell, \lambda) + \ell$. The second part, which is independent of the inputs $\mathbf{x}$ and $\mathbf{y}$, contains pd and $\hat{C}$, and is of size $\text{poly}(\lambda) + |C|$. Overall, the total communication is sublinear to $|\mathbf{x}|$, rate-1 to $|\mathbf{y}|$, and rate-1 to $|C|$.

Alternatively, when we instantiate the batch-MAC with our DCR-based construction (Construction 7), the query size is $O(m^{2/3}\lambda)$. The answer size is $O(m^{2/3}\lambda) + O(\ell^{2/3}\lambda) + \ell + |C| + \text{poly}(\lambda)$.

Theorem 19. *The 2-round 2PC protocol in Construction 18 satisfies correctness, statistical Alice's privacy, and Bob's privacy.*

Proof. We first prove the correctness. By the correctness of the underlying batchMAC, $L_{\mathbf{x}} = K_{\mathbf{x}} + s \cdot \mathbf{x}$ and $L_{\bar{\mathbf{y}}} = K_{\bar{\mathbf{y}}} + s \cdot \bar{\mathbf{y}}$ over $\mathcal{R}$. Thus $z = C(\mathbf{x}, \mathbf{y})$ by the correctness of bitGC.

Note that the query message query is simply a batch-MAC hash digest $d_{\mathbf{x}}$. Thus, the statistical Alice's privacy reduces to the message privacy of batchMAC.

Now we prove Bob's privacy. At a high level, the proof is identical to the privacy proof of the 1-bit-per-gate garbling scheme in [44], except that we call the simulator of the underlying batchMAC to simulate the input labels, instead of sampling randomly. We first recap the following lemmas of [44].

Lemma 13 (Simulatability of GC Bit). *There exists a simulator Sim and a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$, every boolean circuit C with gate g of ℓ_v inputs, every masked input $\bar{\mathbf{v}} \in \{0, 1\}^{\ell_v}$, secret key $s \in \mathcal{D}_{\text{sk}}$, additive shares (over $\mathcal{R}$) $L_{\bar{\mathbf{v}}}, K_{\bar{\mathbf{v}}}$ of $s \cdot \bar{\mathbf{v}}$, and PRF key sk the following holds:*

$$\text{SD}(\text{msg}_G(\text{pd}, K_{\bar{\mathbf{v}}}), \text{Sim}(\text{pd}, L_{\bar{\mathbf{v}}}, \bar{\mathbf{v}}, \bar{w})) \leq \text{negl}(\lambda),$$

where msg_G denotes the 1-bit message that the garbler sends to the evaluator for gate g , and $\text{pd} \xleftarrow{\$} \text{bitGC.Gen}(\text{pp}, s, \text{sk})$, $\bar{\mathbf{v}} = \mathbf{v} \oplus \text{PRF}(\text{sk}, \text{InWires}(g))$, and $\bar{w} = g(\mathbf{v}) \oplus \text{PRF}(\text{sk}, \text{OutWire}(g))$.

Lemma 14 (Simulatability of Public Data). *There exists a simulator Sim such that for every $\text{sk} \in \{0, 1\}^\lambda$, the following two distributions are computationally indistinguishable:*

$$\text{pd} \xleftarrow{\$} \text{Gen}(\text{pp}, s \xleftarrow{\$} \mathcal{D}_{\text{sk}}, \text{sk}), \text{pd}' \xleftarrow{\$} \text{Sim}(1^\lambda, \text{pp}).$$

We describe the simulator required in Definition 12 as follows. It takes $\lambda, \text{pp}, \text{pk}, \mathbf{x}, \mathbf{r}_0, C, z \in \{0, 1\}$ as inputs, and simulates the answer message ans .

- Run the simulator in Lemma 14 and return its output as pd .
- Sample a random bit $\bar{v}^{(i)} \xleftarrow{\$} \{0, 1\}$ as the masked value for each wire i of C , except the input wires corresponding to $\mathbf{x}$.

- Compute $(d_x, \text{st}_U^0) \leftarrow \text{batchMAC.Hash}(\text{pk}, x)$ using randomness r_0 . Compute $(d_{\bar{y}}, \text{st}_U^1) \leftarrow \text{batchMAC.Hash}(\text{pk}, \bar{y})$ and denote the randomness as r . Run the simulator of batchMAC in the definition of key privacy, and denote the outputs as $e_x, e_{\bar{y}}$. Return $e_x, e_{\bar{y}}$ and r .
- Run the batchMAC.Ext algorithm respectively on $(\text{pk}, \text{st}_U^0, e_x)$ and $(\text{pk}, \text{st}_U^1, e_{\bar{y}})$, and take the outputs as input labels L_x and $L_{\bar{y}}$.
- For every gate g in C , in the topological order, for which the input labels $L_v^{(i)}$ and masked value $\bar{v}^{(i)}$ are already defined for all $i \in \text{InWires}(g)$, run the evaluate algorithm of bitGC for gate g to get a label $L_{\bar{w}}^{(j)}$ for the output wire $j \in \text{OutWire}(g)$.
Then, run the simulator in Lemma 13 on $(\text{pd}, (L_v^{(i)}, \bar{v}^{(i)})_{i \in \text{InWires}(g)}, \bar{w}^{(j)})$, where $\bar{w}^{(j)}$ is the masked bit assigned to wire $j = \text{OutWire}(g)$. Return its output bit as the garbled table for g .
- Let $\bar{z}$ be the masked bit assigned to the output wire of C , simulate the mask of the output wire as $\bar{z} \oplus z$.

Now we show a series of hybrid experiments, where Hybrid_0 describes the original experiment and Hybrid_5 describes the simulated experiment.

- Hybrid_0 : The real distribution of $\text{ans} = (\text{ct}_x, \bar{y}, r, \text{ct}_{\bar{y}}, \text{pd}, \hat{C})$, as in Construction 18.
- Hybrid_1 : Instead of computing the 1-bit-per-gate garbled gate in $\hat{C}$ as in bitGC scheme (Lemma 12), simulate them using the simulator in Lemma 13. To be precise, first compute all the wire values $v^{(j)}$ correctly, for each wire $j \in \text{Wire}(C)$. Next compute the masked bit correctly, $\bar{v}^{(j)} = v^{(j)} \oplus \text{PRF}(\text{sk}, j)$, except those inputs grasped by Alice (namely x). Then correctly compute the input labels $L^{(i)} = K^{(i)} + s \cdot \bar{v}^{(i)}$ for each input wire $i \in \text{InWires}(C)$ from the underlying batchMAC.

For every gate g in the topological order, of which input labels $L_v^{(i)}$ and masked value $\bar{v}^{(i)}$ are already computed for all $i \in \text{InWires}(g)$, run the evaluating algorithm of bitGC for gate g to get a label $L_{\bar{w}}^{(j)}$ for the output wire $j \in \text{OutWire}(g)$. Run the simulator in Lemma 13, named Sim_0 , to get a bit,

$$b_g \leftarrow \text{Sim}_0 \left(\text{pd}, (L_v^{(i)}, \bar{v}^{(i)})_{i \in \text{InWires}(g)}, \bar{w}^{(j)} \right),$$

where $\bar{w}^{(j)}$ is the masked bit of wire $j = \text{OutWire}(g)$.

$\text{Hybrid}_1 \stackrel{c}{\approx} \text{Hybrid}_0$ by the correctness of batchMAC and Lemma 13. Note that in Hybrid_1 , the input garbling keys $K^{(i)}$ for $i \in \text{InWires}(C)$, which are derived from batchMAC, are not used in computing the garbled circuit anymore.

- Hybrid_2 : In this hybrid, the input labels $L^{(i)}$ for $i \in \text{InWires}(C)$ are not derived from the underlying batchMAC. Instead, they are simulated by first simulating the compressive authentication e in batchMAC, then locally running the batchMAC.Ext algorithm on the simulated message. To be precise, first compute

$$(d_x, \text{st}^0) \stackrel{\$}{\leftarrow} \text{batchMAC.Hash}(\text{pk}, x), (d_{\bar{y}}, \text{st}^1) \stackrel{\$}{\leftarrow} \text{batchMAC.Hash}(\text{pk}, \bar{y}).$$

Denote the randomness used as $\mathbf{r}_0$ and $\mathbf{r}$, respectively. Then, run the simulator for key privacy of `batchMAC`, named Sim_1 , to simulate the `batchMAC` compressive authentication,

$$e_{\mathbf{x}} \leftarrow \text{Sim}_1(1^\lambda, \text{pk}, \mathbf{x}, \mathbf{r}_0), \quad e_{\bar{\mathbf{y}}} \leftarrow \text{Sim}_1(1^\lambda, \text{pk}, \bar{\mathbf{y}}, \mathbf{r}).$$

Simulate the input labels $L_{\mathbf{x}}$ and $L_{\bar{\mathbf{y}}}$ by computing

$$L_{\mathbf{x}} \leftarrow \text{batchMAC.Ext}(\text{pk}, e_{\mathbf{x}}, \text{st}^0), \quad L_{\bar{\mathbf{y}}} \leftarrow \text{batchMAC.Ext}(\text{pk}, e_{\bar{\mathbf{y}}}, \text{st}^1).$$

$\text{Hybrid}_2 \stackrel{c}{\approx} \text{Hybrid}_1$ by the key privacy of `batchMAC` scheme. Note that in Hybrid_2 , the global secret key s is not used in deriving input labels, and is only used to compute `pd`.

- Hybrid_3 : Instead of computing the mask on the output wire directly using the PRF (namely $\text{PRF}(\text{sk}, \text{OutWire}(C))$), compute it as $\bar{z} \oplus z$, where $\bar{z}$ is the masked wire value on the output wire.

$\text{Hybrid}_3 \approx \text{Hybrid}_2$ by the correctness of the 2PC protocol.

- Hybrid_4 : In this hybrid, instead of correctly compute `pd` as in `bitGC`, simulate it using the simulator in Lemma 14, named Sim_2 ,

$$\text{pd} \leftarrow \text{Sim}_2(1^\lambda, \text{pp}).$$

$\text{Hybrid}_4 \stackrel{c}{\approx} \text{Hybrid}_3$ by Lemma 14. Note that in Hybrid_4 , the PRF key sk is only used to compute the masked wire values, namely $\bar{v}^{(i)}$ for $i \in \text{Wire}(C)$.

- Hybrid_5 : In this hybrid, instead of computing the masked wire values $\bar{v}^{(i)}$ using the PRF, directly sample them randomly as $\bar{v}^{(i)} \stackrel{\$}{\leftarrow} \{0, 1\}$. $\text{Hybrid}_5 \stackrel{c}{\approx} \text{Hybrid}_4$ by the security of PRF. Note that Hybrid_5 is exactly the simulated distribution of `ans`.

We have shown an efficient 2PC protocol by combining our LWE-based `batchMAC` and the lattice-based garbling scheme of [44]. Similarly, we can incorporate our DCR-based `batchMAC` with the Paillier-group-based garbling scheme is [44]. The consequent 2PC protocol enjoys the same merits as the aforementioned lattice-based protocol, except that the communication dependency on $|\mathbf{x}|$ will be $|\mathbf{x}|^{2/3}$ instead of $|\text{poly}(\log |\mathbf{x}|)|$. We first summarize the 1-bit-per-gate garbling scheme of [44] in the following lemma.

Lemma 15 (1-bit-per-gate Garbling under Paillier Groups [44]). *Set the Paillier group public parameters $\text{pp} := (G, F, H, f, N, \ell)$ as in Lemma 1.*

Further, set the garbling keys and labels as follows:

- Sample a global key $s \stackrel{\$}{\leftarrow} [N]$, which is used to generate garbling labels;
- Let $\text{PRF} : \{0, 1\}^\lambda \times \{0, 1\}^\lambda \rightarrow \{0, 1\}$ be a secure PRF in NC1. Sample a PRF key $\text{sk} \stackrel{\$}{\leftarrow} \{0, 1\}^\lambda$;
- The inputs consist of two parts. For the first part $\mathbf{x} \in \{0, 1\}^m$, the evaluator knows $\mathbf{x}$. For the second part $\mathbf{y} \in \{0, 1\}^\ell$, the evaluator acquires its one-time-pad $\bar{\mathbf{y}} := \mathbf{y} \oplus \text{PRF}(\text{sk}, \text{Wire}(\mathbf{y}))$;

- Sample the garbling keys for each input. $K_{\mathbf{x}} \xleftarrow{\$} [N]^m$, $K_{\mathbf{y}} \xleftarrow{\$} [N]^\ell$;
- Set the garbling labels as $L_{\mathbf{x}} = K_{\mathbf{x}} + s \cdot \mathbf{x}$, $L_{\mathbf{y}} = K_{\mathbf{y}} + s \cdot \mathbf{y}$, computed in $\mathbb{Z}$.

Given the above conditions, and assuming the circular power DDH (CP-DDH) assumption in Paillier groups, there exists a garbling scheme. The generation, garbling, and evaluation algorithms are identical to those in Lemma 12:

- $\text{Gen}(\text{pp}, s, \text{sk}) \rightarrow \text{pd}$.
- $\text{Gb}(\text{pd}, s, \text{sk}, K_{\mathbf{x}}, K_{\mathbf{y}}, C) \rightarrow \hat{C}$.
- $\text{Ev}(\text{pd}, L_{\mathbf{x}}, \mathbf{x}, L_{\mathbf{y}}, \mathbf{y}, C, \hat{C}) \rightarrow z$.

The size of pd is $\text{poly}(\lambda)$, and the size of $\hat{C}$ is exactly $|C|$.

The construction is almost identical to Construcion 18. The only difference is that the garbling key s is now a scalar in $\mathbb{Z}$, instead of a polynomial in $\mathbb{Z}[X]/(X^n+1)$ (which can be viewed as a vector in $\mathbb{Z}^n$). Consequently, in Construcion 18, Alice and Bob invoke n parallel `batchMACs` to establish $L_{\mathbf{x}} = K_{\mathbf{x}} + s \cdot \mathbf{x}$ or $L_{\mathbf{y}} = K_{\mathbf{y}} + s \cdot \mathbf{y}$. While in the Paillier-group-based construction, only a single `batchMAC` is needed for $\mathbf{x}$ (the same for $\mathbf{y}$). We leave out the full construction and proofs, and summarize the resulting protocol as the following theorem. Note that the CP-DDH assumption in Paillier groups already implies the DCR assumption.

Theorem 20 (Succinct 2-Round 2PC under Paillier Groups). *Assuming the CP-DDH assumption in Paillier groups, there exists a 2-round 2PC protocol (as defined in Definition 12) satisfying correctness, statistical Alice’s privacy, and Bob’s privacy.*

The communication cost is $O(m^{2/3}\lambda) + O(\ell^{2/3}\lambda) + \ell + |C| + \text{poly}(\lambda)$, where $m = |\mathbf{x}|$ and $\ell = |\mathbf{y}|$. Moreover, the $(|C| + \text{poly}(\lambda))$ -sized part is input-independent and can be generated and transmitted offline.

Remark 4. [44] also proposed an enhanced garbling scheme that only costs 1-bit per lookup gate. We point out that the input labels of this enhanced garbling scheme is exactly the same as the 1-bit-per-gate scheme. Thus by simply altering the underlying garbling scheme ($\text{Gen}, \text{Gb}, \text{Ev}$) to an enhanced one in a black-box way, and without changing our construction, we can decrease the $|C|$ -sized communication to $|C|/\log \log \lambda$ for layered circuits.