Succinct Homomorphic MACs from Groups and Applications*

Yuval Ishai[1], Hanjun Li[2], and Huijia Lin[2]

[1] Technion, Haifa, Israel
yuvali@cs.technion.ac.il
[2] University of Washington, Seattle, WA, USA
{hanjul,rachel}@cs.washington.edu

Abstract. Homomorphic message authentication codes (HMACs) allow users to authenticate data using a shared secret key, while supporting computation over authenticated data. Given data $(m_1, \ldots, m_n)$ and their tags $(\sigma_1, \ldots, \sigma_n)$, anyone can evaluate a circuit C on the data and tags to produce a succinct tag authenticating the output $C(m_1, \ldots, m_n)$. Importantly, tags remain *succinct*—of size polynomial in the security parameter λ—regardless of the size of C. This work introduces an enhanced variant of HMACs called *algebraic HMAC* (aHMAC), in which all tags (input and output) take the form $\boldsymbol{\Delta} \cdot m + \boldsymbol{K}$, as in standard information-theoretic MACs.

We construct an aHMAC from group-based assumptions, including variants of the DDH and DCR assumptions, and use it to obtain group-based constructions of several cryptographic primitives:

- **Succinct CDS for circuits.** For any $P : [N]^k \to [N]$ represented by circuit, we obtain a Conditional Disclosure of Secrets protocol with $\mathrm{poly}(\lambda, k, \log N)$ communication.
- **Succinct PSM for simple programs.** For any $P : [N]^k \to [N]$ represented by a truth-table or shallow branching program, we obtain a Private Simultaneous Messages protocol or a garbling scheme with $\mathrm{poly}(\lambda, k, \log N)$ communication.
- **Constrained PRFs for circuits.** We obtain the first group-based constrained pseudorandom functions for general circuits, improving over a previous construction for NC^1 circuits.

Prior to our work, these applications could only be obtained from lattice assumptions or indistinguishability obfuscation.

* This is an updated version of [ILL24], with the old title "Succinct Partial Garbling from Groups and Applications." See Section 1.7 for a summary of the changes.

Table of Contents

1 Introduction

Diversifying assumptions is a central theme in cryptography. While almost all cryptographic primitives can be built from powerful tools such as indistinguishability obfuscation (iO), in most cases this is an overkill. Excluding iO, lattice-based assumptions have also shown great versatility in implying a wide range of primitives. For many of these primitives, we still do not have constructions from cryptographic groups or other number-theoretic assumptions. In particular, primitives with homomorphic evaluation capabilities, such as homomorphic encryption, attribute-based encryption, homomorphic MACs/signatures, and succinct garbling schemes have state-of-the-art constructions using lattices, while group-based or number-theoretic constructions typically only suffice for more limited functionalities or efficiency features.

This work is motivated by the challenge of narrowing the gap between the provable consequences of iO or lattices and group-based assumptions. Besides the direct benefit of diversifying assumptions, efforts in this direction have often led to other major developments and inspired new techniques that found unexpected applications.

Expanding group-based cryptography. In this work, we provide the first group-based constructions for central cryptographic primitives that were previously only known using iO or lattices. This includes the following:

- Succinct protocols for Conditional Disclosure of Secrets (CDS) [GIKM00] for general circuits, improving over a previous one-way function based construction for truth tables [ABI+23];
- Succinct protocols for Private Simultaneous Messages (PSM) [FKN94] and garbling schemes [Yao86] for truth tables and shallow branching programs, providing the first fully succinct group-based constructions for any kind of programs;
- Constrained Pseudo-Random Functions (CPRF) [BW13, KPTZ13, BGI14] for general constraint circuits, overcoming the NC^1 barrier in previous group-based constructions [CMPR23].

For all these primitives, we present the first group-based constructions, under (variants of) standard assumptions. Our new constructions are enabled using Homomorphic MAC (HMAC) with natural and useful algebraic properties – called algebraic HMAC (aHMAC) – that we construct using groups. Along the way, we also significantly improve over previous group-based HMAC schemes without the algebraic property, in particular, removing depth constraint, dispensing the need for sub-exponential hardness and non-black-box usage of cryptography. Let us explain more.

Main Tool: Algebraic Homomorphic MAC. A standard homomorphic MAC, introduced by Gennaro and Wichs [GW13] following Agrawal and Boneh [AB09], enables users to authenticate their data using a (shared) secret key. After receiving data $(m_1, \cdots, m_n)$ together with their MAC tags $(\sigma_1, \cdots, \sigma_n)$, anyone can homomorphically execute a program P over the data and tags to generate a succinct tag that authenticates the output of $P(m_1, \cdots, m_n)$; verification of the output tag can be done without even knowing the original inputs. Similar to homomorphic encryption, HMAC is *composable* in the sense that one can incrementally combine authenticated outputs of partial computations to derive an authenticated output of a larger computation. The key feature of homomorphic MAC is that the tags are *succinct*, of size $\mathrm{poly}(\lambda)$, independent of the computation complexity or even the input length. As such, they enable verifying computations over outsourced data in a communication-succinct way (though verification may take as long as the verified computation).

In this work, we propose an algebraic enhancement, called aHMAC, requiring that the tags of aHMAC (both input and evaluated ones) have the most widely used algebraic form $\mathbf{\Delta} \cdot m + \mathbf{K}$,

as in a standard information-theoretic MAC. Here $\mathbf{\Delta}$ is a λ-bit global authentication key and K is a pseudorandom blinding term, that can be derived from the secret MAC key. In particular, given input tags $\{\mathbf{\Delta} \cdot x_i + \mathbf{K}_i\}_i$ and the inputs $\{x_i\}$, homomorphic evaluation of a function f yields output tag $\mathbf{\Delta} \cdot y + \mathbf{K}_f$, where $y = f(\{x_i\})$, and the key $\mathbf{K}_f$ can be computed from the original keys $\{\mathbf{K}_i\}$ according to the function f, independent of the inputs $\{x_i\}$. We show that aHMAC readily implies HMAC (see Section 3.5), and the additional algebraic properties are very useful for applications.

Constructions of HMAC: There have been a large body of works on constructing HMAC and its stronger public key variant homomorphic signatures [BF11]; see Table 1 (and references in [ACG24]). Not surprisingly (by now), using lattice assumptions or iO together with other standard tools, one can obtain fully composable homomorphic MACs and signatures for all polynomial-sized circuits with succinct tags/signatures [GW13, GVW15b, GU24]. On the "low end," assuming just one-way functions, partially succinct HMACs are known for computing low degree polynomials [CF13], which can further be made fully succinct using groups [CF13, BFR13, CFGN14]. In a more recent work [ACG24], using non-interactive Batch Arguments (BARGs), one can even obtain succinct HMAC for bounded-depth circuits assuming subexponential k-Lin over pairing groups or DDH over non-pairing groups. However, the way BARG is used makes the construction non-black-box and inefficient. The weakness of supporting only bounded-depth circuits and reliance on sub-exponential hardness and non-black-box usage of cryptography highlight gaps between the best known group-based HMAC and constructions using lattices or iO. Another line of works [CFT22, KLVW23, WW24], leveraging pairing-group, circumvents these drawbacks but does not achieve the full functionality of HMAC, since they only permit very limited composability. This leaves open the following questions:

> *Is there a black-box construction of (non-algebraic) HMACs for circuits from polynomial hardness of standard group assumptions? Or any construction of aHMAC from groups?*

In this work, we answer both questions affirmatively for the stronger notion of aHMAC. More specifically, we present *black-box* constructions of 1) aHMAC for bounded depth circuits based on DDH in prime-order or Paillier groups, or DCR in Paillier groups [Pai99, DJ01], and 2) aHMAC for all circuits, based on their circular variants. This immediately implies similar HMAC constructions under the same assumptions. These constructions can also be instantiated using prime-order groups, assuming DDH for bounded depth circuits or circular power-DDH for all circuits, albeit with (an arbitrarily small) inverse-polynomial error that can be eliminated in the context of applications.

Applications to CDS, PSM, and CPRF. Leveraging the algebraic property of our aHMAC, we are able to significantly extend the reach of group-based assumptions, obtaining succinct CDS for circuits, succinct PSM for truth tables and other simple programs, as well as CPRF for circuits, from the same assumptions enabling our aHMAC. All of these primitives were previously only known under lattices or iO.

1.1 Applications

We now give a more detailed account of the above applications.

Conditional Disclosure of Secrets. A k-party Conditional Disclosure of Secret (CDS) protocol [GIKM00] considers a setting where k parties want to disclose a common secret $s \in \{0,1\}$ to a referee if and only if their respective inputs $x_1, \cdots, x_k$ (known to the referee) satisfy some

	Class	$\lvert\sigma\rvert$	Assumption	BBox	Composable
HSig [GU24, GVW15b]	Cir	$\mathrm{poly}(\lambda)$	subexp iO + FHE + NIZK or SNARK-NP	✗	✓
HMAC [GW13]	Cir	$\mathrm{poly}(\lambda)$	FHE	✓	✓
HSig [GVW15b]	Bnd-Dep-Cir	$\mathrm{poly}(\mathrm{Dep}, \lambda)$	SIS	✓	✓
HSig [ACG24]	Bnd-Dep-Cir	$\mathrm{poly}(\mathrm{Dep}, \lambda)$	subexp k-Lin (Pairing) or subexp LWE or subexp DDH	✗	✓
HMAC [CF13]	poly deg polynomials	$\frac{\mathrm{Deg} \cdot \mathrm{poly}(\lambda)}{\mathrm{poly}(\lambda)^\bigstar}$	$\frac{\mathrm{OWF}}{\mathrm{DDH}}$	✓	✓
aHMAC [This Work]	Cir	$\mathrm{poly}(\lambda)$	CP-DDH or KDM-DCR	✓	✓
aHMAC [This Work]	Bnd-Dep Cir	$\mathrm{Dep} \cdot \mathrm{poly}(\lambda)$	DDH	✓	✓
Schemes with Limited Composability					
HSig [CFT22]	NC^1	$\mathrm{poly}(\lambda)$	DHE	✓	✗
HSig [KLVW23]	Cir	$\mathrm{poly}(\lambda)$	k-Lin (Pairing) or LWE or subexp DDH	✗	✗
HSig [WW24]	bounded size Cir	$\mathrm{poly}(\lambda)$	bilateral k-Lin (Pairing)	✓	✗

Table 1. Comparison between homomorphic MAC and signature schemes, in terms of *class of functions supported*, size $|\sigma|$ of the tags/signatures, *assumptions*, whether the construction make *black-box* use of the underlying cryptographic tools and assumptions, and whether the scheme is fully *composable*. The top part lists schemes that are composable, while the bottom part lists weaker schemes that are not. λ denotes the security parameter, Dep the depth of a circuit, and Deg the degree of a polynomial. DHE stands for the Diffie-Hellman exponent assumption. $^\bigstar$ indicates that the computational costs of the DDH-based HMAC scheme of [CF13] scales with the degree of the polynomial evaluated.

fixed (public) predicate $P : [N]^k \to \{0, 1\}$. Using randomness, the parties achieve this by sending simultaneous messages to the referee. This notion is equivalent to secret sharing for partite functions and partial garbling schemes [IW14]. CDS has found many interesting applications, including to symmetric private information retrieval protocols [GIKM00], attribute based encryption [GVW15a, Att14, Wee14, IW14], (priced) oblivious transfer [AIR01], and secret sharing [LV18, ABF$^+$19, ABNP20, AN21, ABI$^+$23].

Understanding the minimal communication complexity of CDS has been an active research direction. This question is qualitatively interesting because without security requirements, one bit of communication suffices by simply sending the secret to the referee. This motivates the challenge of *fully succinct CDS*, where the total communication is independent of the description size of the predicate P. For the simplest case of a *truth-table* representation of P, in the information theoretic two-party setting, partially succinct schemes are known with communication $2^{O(\sqrt{n \log n})}$ [LVW18], where $n = \log N$. In the computational setting, it is known that one-way functions are sufficient for $\mathrm{poly}(\lambda, k, \log N)$ communication (poly-logarithmic in the truth-table size N^k) [HK20, ABI$^+$23, Hea24], but again with computational cost that scales polynomially with the input domain size, or equivalently the truth-table representation. For general predicates P expressed as Boolean circuits over the bit-representation of the inputs, succinct and efficient CDS is only known using lattices or iO [BGG$^+$14, HLL23].

We present the first group-based construction of succinct CDS for circuits based on the same assumptions underlying our aHMAC. Interestingly, while aHMACs instantiated using prime-

order groups have inverse-polynomial errors, these errors can be eliminated in CDS through correctness and security amplification, giving fully correct and secure instantiation from prime-order groups.

Private Simultaneous Messages. A Private Simultaneous Messages (PSM) protocol [FKN94, IK97] enables k parties with shared common randomness to securely evaluate a public predicate $P : [N]^k \to \{0,1\}$ on their *private* inputs $x_1, \cdots, x_k$, by simultaneously sending messages to a referee, revealing only the output $P(x_1, \ldots, x_k)$. For $N = 2$, it is equivalent to decomposable randomized encoding [IK00] and garbling schemes [Yao86, BHR12]. Similar to CDS, the PSM model has been a test bed for understanding the communication overhead of secure computation in a simple non-interactive setting, as well as for yielding secure computation protocols with efficient online communication [AIKW13].

The additional requirement of hiding the inputs from the referee makes PSM more challenging to realize than CDS. Even for the simplest truth-table representation of P, the only known succinct constructions uses lattice [GKP$^+$13, BGG$^+$14, HLL23] or iO [KLW15, BCG$^+$18]. Without lattices or iO, the best known protocols are information theoretic and have communication complexity grows polynomially with N: $O(N^{1/2})$ for $k = 2$ [BIKK14] and $O_k(N^{(k-1)/2})$ for infinitely many k [BKN18, AL19]. We obtain a (computationally secure) PSM protocol and garbling schemes with communication complexity $\mathrm{poly}(\lambda, k, \log N)$, polylogarithmic in the truth-table size N^k and computation complexity $\mathrm{poly}(\lambda, N^k)$, based on DDH in prime order groups, or DCR in Paillier groups. This yields exponential improvement in communication compared to the previous state-of-the-art without using lattices or iO. More generally, our PSM protocol can handle functions P represented by shallow branching programs, including DFAs and decision trees, with computational cost that scales polynomially with the program size.

Constrained PRFs. Constrained pseudorandom function (CPRFs) [BW13, KPTZ13, BGI14] allow for delegating "constrained" secret keys that enable evaluating the function at certain authorized inputs – as specified by a constraint predicate $P : \{0,1\}^n \to \{0,1\}$ – while keeping the function value at unauthorized inputs pseudorandom. Here we consider the common scenario where a single constrained key is published for a selectively chosen function–referred to as 1-key selective CPRF. The stronger variants supporting multiple keys and/or achieving adaptive security are only known using heavy tools such as iO.

Assuming one-way functions, the works of [BW13, KPTZ13, BGI14] showed how to construct CPRFs for simple function classes that include prefix-fixing functions and range functions, which through the "puncturing" technique of Sahai and Waters [SW14] become a crucial tool in many iO applications. CPRFs for general functions have other applications, such as, broadcast encryption schemes and identity-based key exchange [BW13].

We obtain CPRFs for general constraint circuits, based on DCR, or DDH plus the small exponent assumption in Paillier groups. Previously, CPRFs for general circuits were only known relying either on lattices [BV15] or multilinear maps [BW13] or iO [HKKW19, BLW17, DKN$^+$20] plus other tools. Group-based CPRFs were limited to NC^1 circuits [GGM84, AMN$^+$18, DKN$^+$20, CMPR23], and our new construction overcomes this NC^1 barrier.

In summary, in this work, we significantly extend the reach of group-based cryptography, by providing the first group-based constructions of several important primitives including succinct CDS for circuits, PSM for truth table, and CPRFs for circuits. We obtain these results through the new aHMAC primitive, an algebraic enhancement of HMAC that is likely to find other applications. Our group-based construction of aHMAC also represents significant improvement

over previous group-based HMACs, dispensing with depth constraint, sub-exponential hardness, and non-black-box use of cryptography.

In the body of the paper, we present our succinct PSM and CDS protocols using a unified framework of *garbling schemes* and *partial garbling* schemes, respectively.

1.2 Overview of aHMAC Construction

Our Assumptions. The set of assumptions we use in this work includes the standard assumption of DDH in prime-order groups and its adaptation in Paillier groups, DCR in Paillier groups, as well as their circular security variants. We assume familiarity with the DDH assumptions and give a short description of the circular variants of DDH and DCR.

The Circular Power Decisional Diffie Hellman (CP-DDH) assumption (Definition 9) asserts that a circular encryption of bits of the secret key s using powers of the secret key is pseudorandom. More precisely, for appropriately sampled group elements g, f and random exponents s and $\{a_i, b_i, c_i\}_i$, the following computational indistinguishability holds:

$$g, g^s, (g^{a_i}, g^{sa_i}, g^{s^2 a_i} \cdot f^{s[i]})_{i\in[\log s]} \qquad \approx_c \qquad g, g^s, (g^{a_i}, g^{b_i}, g^{c_i})_{i\in[\log s]}.$$

The CP-DDH assumption can be postulated over prime-order groups, as well as the Paillier/Damgård-Jurik and class groups, and our constructions can be instantiated using any of these groups. For prime-order groups, CP-DDH holds in the standard generic group model (GGM) [Sho97]; see Section 2.4. In particular, our CDS and succinct partial garbling scheme is unconditionally secure in the GGM.

Alternatively, we can replace CP-DDH in the Paillier/Damgård-Jurik groups by assuming key-dependent message (KDM) security of the Damgård-Jurik cryptosystem (Definition 11) [BRS03, BHHO08]. The semantic security of this cryptosystem relies on the Decisional Composite Residuosity (DCR) assumption [DJ01]. KDM variants of this assumption are commonly used in prior Paillier-group based HSS works, for example [FGJS17, OSY21, MORS24].[3] We write KDM-DCR as a shorthand.

Theorem 1 (Algebraic Homomorphic MAC, Informal; see Theorem 5, 6). *Assume DDH over prime-order or Paillier groups. There is an aHMAC (with tags of the form $\mathbf{\Delta} \cdot y + \mathbf{K}$) for bounded-depth circuits, where the evaluation key size scales linearly with the circuit depth.*

Under the CP-DDH assumption over prime-order or Paillier groups, or alternatively KDM-DCR over Paillier groups, the aHMAC can support general circuits with fixed $\mathrm{poly}(\lambda)$*-size evaluation key.*

Instantiations using prime-order groups have inverse polynomial $1/poly(\lambda)$ correctness errors, while instantiations using Paillier groups have negligible correctness errors.

Technical Overview We provide an overview of our aHMAC construction, which builds heavily on techniques from the homomorphic secret sharing (HSS) literature. HSS [BGI16] is a 2-party analogue of FHE, allowing a local mapping from shares of an input x to *additive* shares of an output $f(x)$. However, group-based HSS cannot be applied directly to construct an aHMAC for two reasons: *i)* known group-based HSS schemes only support limited classes of circuits captured

[3] More concretely, our notion of KDM security assumes that encryptions (under sk) of sk^{-1} are secure. This is strictly weaker than [MORS24], which assumes encryptions of any affine function of $\mathsf{sk}, \mathsf{sk}^{-1}$ are secure, and incomparable with [FGJS17, OSY21], which assume encryptions of bits of sk are secure.

by so-called "RMS programs" and *ii)* in standard group-based HSS schemes, *both* shares depend on the input. In aHMAC, there is an "asymmetry of information": The homomorphic evaluation over tags can depend on inputs to compute the output tag $\Delta \cdot y + \mathbf{K}_f$, but not the derivation of $\mathbf{K}_f$ from the original keys $\{\mathbf{K}_i\}$ (which only depends on f).

The HSS limitation to RMS programs stems from the fact that if two parties hold only additive shares of two values x and y, it is not clear how to "multiply" these shares to obtain shares of the product xy. A key idea in HSS is that if x is an input and the two parties additionally have an appropriate encryption $\mathsf{Enc}_{\mathbf{s}}(x \cdot \mathbf{s})$ of x multiplied by the secret key $\mathbf{s}$, x can be "multiplied" with shares of $(y \cdot \mathbf{s})$ to obtain shares of $(xy \cdot \mathbf{s})$. As a result, the computation of both parties depends on x.

The difficulty with handling general circuits is that there is no encryption for intermediate computation values, only shares of $(x \cdot \mathbf{s})$ and $(y \cdot \mathbf{s})$. A key technical contribution of this work is a method for "multiplying" these shares, when x and y are known to just one party (e.g., the evaluator) while the computation of the other party (e.g., the authenticator) is independent of x, y. Our technique is inspired by the recent work of Meyer, Orlandi, Roy, and Scholl [MORS24], but with crucial differences. The work of [MORS24] designed a way of "multiplying" shares of $(x \cdot \mathbf{s})$ and $(y \cdot \mathbf{s})$, without knowledge of x, y, by revealing to one party auxiliary information that depends on the other party's shares. Our technique eliminate the need for auxiliary information, by leveraging that x, y are known to one party.

We now describe our aHMAC based on the CP-DDH assumption that supports all circuits. This construction can be easily modified to obtain a leveled variant for bounded depth circuits, from power-DDH (without circular security). To obtain a construction based on standard DDH, we rely on a technique from the recent work [MORS25].

More technically, in aHMAC, a KeyGen algorithm generates a secret key sk and an evaluation key evk. They are distributed to an authenticator and an evaluator respectively. The authenticator can use the secret key sk to compute tags σ_x for inputs x (with an arbitrary unique id). An evaluator can use the evaluation key evk to homomorphically evaluate any Boolean circuit C over the tags σ_x.

$$\begin{array}{ll} \underline{\text{Authenticator}}(\mathsf{sk}): & \underline{\text{Evaluator}}(\mathsf{evk}): \\ \sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id}), & \sigma_z \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \{\sigma_x^{(i)}\}). \end{array}$$

Correctness requires the evaluated tag σ_z to have an algebraic form (hence the name) $\sigma_z = \Delta \cdot z + k_C$ over $\mathbb{Z}$, where $z = C(\mathbf{x})$, Δ is a global secret specified at key generation time, and k_C is a MAC key that can be derived without knowing the authenticated inputs $\mathbf{x}$:

$$k_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \{\mathsf{id}^{(i)}\}), \text{// s.t. } \sigma_z = \Delta \cdot C(x) + k_C \text{ over } \mathbb{Z}.$$

Security requires the global secret Δ remains hidden to the evaluator. Succinctness requires the tags, including the evaluated ones, have bit-lengths independent of the evaluation circuit C.

Constructing aHMACs from DDLog. The authentication algorithm $\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$ is simple. It evaluates a PRF on the id to produce a one-time pad $k_x \leftarrow \mathsf{F}(\mathsf{key}, \mathsf{id})$, and outputs $\sigma_x := \Delta \cdot x + k_x$ over $\mathbb{Z}$. The PRF key key and the global secret Δ are included in sk.

The core of our construction is an evaluation procedure to derive from two algebraic tags σ_x, σ_y to another σ_z while maintaining their algebraic forms:

$$\begin{array}{ll} \underline{\text{Authenticator}}(\mathsf{sk}): & \underline{\text{Evaluator}}(\mathsf{evk}): \\ \text{from } k_x, k_y & \text{from } \sigma_x = \Delta \cdot x + k_x,\ \sigma_y = \Delta \cdot y + k_y, \\ \text{to } k_z, & \text{to } \sigma_z = \Delta \cdot z + k_z, \end{array}$$

for the cases $z = x + y$ and $z = x \cdot y$ over $\mathbb{Z}$. This would give us the algorithms $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$ respectively for evaluating circuits consisting of Add and Mult gates. As we will see, our evaluation procedure requires the input values x, y to be polynomially bounded. But this still suffices for evaluating any Boolean circuit, which can be implemented via Add, Mult gates while keeping intermediate values bounded by 1.

We first note the easy case, $z = x+y$: setting $\sigma_z := \sigma_x + \sigma_y$, and $k_z := k_x + k_y$ over $\mathbb{Z}$ suffices. We focus on the case of $z = x \cdot y$. Our starting point is the following identity

$$\sigma_x \cdot \sigma_y - \Delta(x \cdot \sigma_y + y \cdot \sigma_x) + (\Delta^2 + \Delta)z = \Delta \cdot z + k_x \cdot k_y,$$

where the RHS would be a tag for z of the desired form, with $k_z = k_x \cdot k_y$. While the terms $\sigma_x \cdot \sigma_y$, $(x \cdot \sigma_y + y \cdot \sigma_x)$, and z are computable by the evaluator, the apparent challenge is to allow the evaluator to multiply Δ, and $(\Delta^2 + \Delta)$ with those terms without leaking Δ. For this, we apply the natural idea of putting Δ in the exponents of a random group (with a generator g) element h, and compute the identity *in the exponents*:

$$\begin{aligned} & r \leftarrow \$, \quad h = g^r, \quad h_1 = h^{\Delta}, \quad h_2 = h^{\Delta^2} g^{\Delta}, \\ \implies & h^{\sigma_x \cdot \sigma_y} / h_1^{x \cdot \sigma_y + y \cdot \sigma_x} \cdot h_2^z = g^{\Delta \cdot z} \cdot h^{k_x \cdot k_y}. \end{aligned} \tag{1}$$

The evaluation key evk consists exactly of those group elements h, h_1, h_2. By a DDH-like assumption, which we call circular-power-DDH (CP-DDH), the terms h, h_1, h_2 together don't leak Δ.

It remains to recover the exponents into an integer satisfying the desired algebraic from. The rich literature of HSS constructions provides two methods. (a) The work of [BGI16, DKK18] present an algorithm, DDLog that works for any cyclic group (of order p) and small exponents $m < \text{poly}$, but fails with $1/\text{poly}$ probability over the choice of a common public randomness R:

$$\forall a \in \langle g \rangle, \quad \Pr_R[\mathsf{DDLog}_g(a \cdot g^m; R) = m + \mathsf{DDLog}_g(a; R) \bmod p] \geq 1 - 1/\text{poly}. \tag{2}$$

This will lead to an aHMAC scheme with overall $1/\text{poly}$ correctness error. (b) The work of [ADOS22] introduces a framework of groups (including the Damgård-Jurik groups and class groups) with efficient and perfectly correct DDLog algorithms for certain "easy" subgroups. This will lead to an aHMAC scheme with perfect correctness.

For simplicity, we assume method (a) in this overview. We obtain the following almost correct evaluation procedures:

$$\begin{array}{ll} \underline{\text{Authenticator}}(\mathsf{sk} \ni (h, R), k_x, k_y): & \underline{\text{Evaluator}}(\mathsf{evk} = (h, h_1, h_2, R), \sigma_x, \sigma_y): \\ k_z^* \leftarrow \mathsf{DDLog}_g(h^{k_x \cdot k_y}; R), & a := h^{\sigma_x \cdot \sigma_y} / h_1^{x \cdot \sigma_y + y \cdot \sigma_x} \cdot h_2^z, \\ & \sigma_z^* \leftarrow \mathsf{DDLog}_g(a; R). \end{array}$$

Assuming the term $\Delta \cdot z$ is small, we can apply the DDLog guarantee (although with $1/\text{poly}$ failure probability):

$$\begin{aligned} \sigma^* &= \mathsf{DDLog}_g(a; R) \\ (\text{Eq } 1) &= \mathsf{DDLog}_g(g^{\Delta \cdot z} \cdot h^{k_x \cdot k_y}; R) \\ (\text{Eq } 2) &\equiv \Delta \cdot z + \mathsf{DDLog}_g(h^{k_x \cdot k_y}; R) \equiv \Delta \cdot z + k_z^* \bmod p. \end{aligned}$$

The procedure as described has two more challenges to resolve:

– The DDLog algorithm from (a) requires $\Delta \cdot z$ to be polynomially bounded, while the global secret Δ needs to be exponentially large in CP-DDH. We resolve this by decomposing the large secret Δ into a bit-vector, so that each entry is small. (See Section 3 for details.)
– The DDLog algorithm only ensures $\sigma_z^* = \Delta \cdot z + k_z^* \bmod p$, while we need the equality to hold over $\mathbb{Z}$. A standard trick is to add a common random shift to both σ_z^* and $k_z^* \bmod p$. If p is sufficiently larger than $\Delta \cdot z$, then the equality holds over $\mathbb{Z}$ except with negligible probability. The common random shifts and the randomness R for DDLog can be derived from a public PRF seed included in both sk and evk.

What we obtain now is an aHMAC scheme with 1/poly correctness, and where the global secret Δ is sampled as a secret group exponent. In our applications, it will be convenient if the final evaluated tags σ_z can have a user-supplied global secret Δ' instead. Furthermore, for evaluating a multi-output circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$, we would like the user-supplied secrets to be different for each output bit of $\mathbf{z} = C(\mathbf{x})$. [4] We implement those features in our final constructions in Section 3, which also contains a construction with negl correctness error based on method (b), a construction based on the KDM security of Damgård-Jurik encryption, and leveled variants of the constructions.

We summarize the syntax of our final constructions.

$$
\begin{array}{ll}
(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta}) & \\
\underline{\text{Authenticator}}(\mathsf{sk}) : & \underline{\text{Evaluator}}(\mathsf{evk}) : \\
\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id}), & \sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \{\sigma_x^{(i)}\}), \\
\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \{\mathsf{id}^{(i)}\}), & \quad // \text{ s.t. } \sigma_{\mathbf{z}} = \boldsymbol{\Delta} \odot \mathbf{z} + \mathbf{k}_C \text{ over } \mathbb{Z},
\end{array}
$$

where $\mathbf{z} = C(\mathbf{x})$, and $\odot$ denotes component-wise multiplication between vectors.

1.3 Overview of Succinct Partial Garbling and CDS

Using aHMAC, we first obtain a succinct partial garbling scheme for general circuits, which implies succinct CDS for circuits.

Garbling schemes [Yao86] play a critical role in modern cryptography, enabling non-interactive secure computation in a variety of applications. A garbling scheme is a randomized algorithm transforming a "program" $C : \{0,1\}^n \to \{0,1\}^m$, such as a circuit or a branching program, into a garbled program $\hat{C}$ along with a pair of short keys $(k_{i,0}, k_{i,1})$ for each input bit x_i. Given the program C, the garbled program $\hat{C}$ and the input keys $k_x = (k_{i,x_i})_{i \in [n]}$ for a private input x, anyone can compute $C(x)$ while learning nothing else about x, in the sense that $(\hat{C}, k_x)$ can be simulated (up to computational indistinguishability) given C and $C(x)$ alone.

Motivated by potential efficiency benefits in applications, Ishai and Wee [IW14] extended this notion to *partial garbling*, where part of the input is public. In partial garbling, the program is decomposed into a public part $C_{\text{pub}}(x)$, which depends only on a public input x, and a private part $C_{\text{priv}}(C_{\text{pub}}(x), y)$, which also involves a private input y. Partial garbling generalizes standard garbling, privacy-free garbling [FNO15], and CDS [GIKM00, AARV17]. The latter can be view as a special instance of partial garbling, which in turn also implies partial garbling when combined with standard garbling; see Section 1.3.

[4] This can be trivially achieved by running the single-output scheme ℓ_z times. But in the multi-output version, we require the tags sizes to be independent of ℓ_z.

Yao's original garbling scheme for circuits [Yao86] and its optimization [BMR90, NPS99, KS08, PSSW09, KMR14, GLNP15, ZRE15, RR21] have garbled circuit size growing linearly with the size $|C|$ of the original circuit, creating a communication bottleneck for large-scale computations. *Succinct garbling*, where the size of the garbled program $\hat{C}$ does not grow with C, holds the promise to resolve the bottleneck. Specifically, the garbled circuit size $|\hat{C}|$ should only depend (polynomially) on the security parameter, and possibly also on second-order parameters such as the input and output length. However, existing succinct garbling schemes rely on "high-end" primitives, such as iO [KLW15, BCG+18] or combinations of FHE and ABE [GKP+13, BGG+14, HLL23]. Without iO or lattices, even for the weaker notion of partial garbling (or CDS), succinct schemes where the garbled circuit size is only independent of the size $|C_{\text{pub}}|$ of the public computation, are only known for highly restricted program classes, such as truth tables [HK21, Hea24, ABI+23].

Theorem 2 (Succinct Partial Garbling for Circuits, Informal; see Theorems 9, 10). *Assume DDH over prime-order groups, or Paillier groups, or class groups. Let $\mathcal{C}$ be the class of two-input circuits of the form $C(x, y) = C_{\text{priv}}(C_{\text{pub}}(x), y)$. Then, there is a succinct partial garbling scheme for $\mathcal{C}$ with garbled circuit of size $(|C_{\text{priv}}| + D_{\text{pub}}) \cdot \text{poly}(\lambda)$, where D_{pub} is the depth of the public circuit C_{pub}. Under the CP-DDH assumption, or alternatively KDM-DCR, the garbled circuit size is reduced to $|C_{\text{priv}}| \cdot \text{poly}(\lambda)$.*

Technical Overview As a first step, we compose an aHMAC scheme with any symmetric encryption E to *succinctly* implement a simple case of partial garbling $\mathsf{CDS}(\mathbf{x}, y) = f(\mathbf{x}) \cdot y$.

At high level, the garbler use aHMAC tags for $\mathbf{x}$ as their labels, and prepares an encryption of y under the global secret Δ. To help decryption by the evaluator, the garbler also releases an evaluated MAC key k_f.

$\underline{\mathsf{CDSGb}}(f, y)$:	$\underline{\mathsf{CDSEv}}(f, \mathsf{gb}, \mathbf{x}, \{L_x^{(i)} = \sigma_{\mathbf{x}[i]}^{(i)}\})$:
$\Delta \leftarrow \$,\ (\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \Delta)$	Output 0 if $f(\mathbf{x}) = 0$. O/w:
$\sigma_b^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, b, \mathsf{id}^{(i)})$,	$\sigma_z \leftarrow \mathsf{EvalTag}(\mathsf{evk}, f, \mathbf{x}, \{\sigma_{\mathbf{x}[i]}^{(i)}\})$,
$k_f \leftarrow \mathsf{EvalKey}(\mathsf{sk}, f, \{\mathsf{id}^{(i)}\})$,	$\Delta = \sigma_z - k_f$,
$\mathsf{ct}_y \leftarrow \mathsf{E.Enc}(\Delta, y)$,	$y = \mathsf{E.Dec}(\Delta, \mathsf{ct}_y)$,
Output $\{L_{x,b}^{(i)} = \sigma_b^{(i)}\}$, and $\mathsf{gb} = (\mathsf{evk}, k_f, \mathsf{ct}_y)$.	Output y.

The evaluator, given aHMAC tags for $\mathbf{x}$ can evaluate f on them to obtain a tag $\sigma_z = \Delta \cdot f(\mathbf{x}) + k_f$. (For simplicity, we assume the aHMAC scheme has negl correctness errors in this overview. See Section 4.2 for how to handle 1/poly correctness errors.) When $f(\mathbf{x}) = 1$, the evaluators use k_f to recover Δ, and decrypts y correctly. When $f(\mathbf{x}) = 0$, $k_f = \sigma_z$ is completely redundant. The aHMAC security then guarantees that Δ remains hidden to the evaluator, and hence the ciphertext ct_y under Δ securely hides y.

We can extend the above construction to a slightly more general case, where $\mathsf{CDS}(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) \odot \mathbf{y}$. Here $f : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$ is a multi-output function, and $\odot$ denotes component-wise multiplication. An evaluator can learn the j-th components of the secret input $\mathbf{y}$ if and only if $f(\mathbf{x})[j] = 1$.

Finally, we compose the simple-case succinct partial garbling $\mathsf{CDSGb}, \mathsf{CSDEv}$ with any standard Boolean garbling $\mathsf{BG.Garb}, \mathsf{BG.Eval}$ to handle general cases, $C(\mathbf{x}, \mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}), \mathbf{y})$,

where the public computation P_{pub} outputs intermediate values $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$, and the private computation C_{priv} outputs final results $\mathbf{z} = C_{\text{priv}}(\mathbf{w}, \mathbf{y})$.

At a high level, the construction runs BG to garble the private computation C_{priv}, producing labels for possible values of $\mathbf{w}$ and $\mathbf{y}$, and runs $\mathsf{CDSGb}, \mathsf{CDSEv}$ to release only the labels corresponding to $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$.

$\underline{\mathsf{Garb}}(C = (C_{\text{pub}}, C_{\text{priv}})) :$
$(\widehat{C_{\text{priv}}}, \{L^{(i)}_{w,b}\}, \{L^{(i)}_{y,b}\}) \leftarrow \mathsf{BG.Garb}(C_{\text{priv}}),$
$(\{L^{(i)}_{x,b}\}, \mathsf{gb}) \leftarrow \mathsf{CDSGb}(C_{\text{pub}}, \{L^{(j)}_{w,1}\})),$
$(\{\overline{L}^{(i)}_{x,b}\}, \overline{\mathsf{gb}}) \leftarrow \mathsf{CDSGb}(\overline{C_{\text{pub}}}, \{L^{(i)}_{w,0}\})),$
Output $\{L^{(i)}_{x,b}, \overline{L}^{(i)}_{x,b}\}, \{L^{(i)}_{y,b}\},$
and $\widehat{C} = (\widehat{C_{\text{priv}}}, \mathsf{gb}, \overline{\mathsf{gb}})$.

$\underline{\mathsf{Eval}}(C, \widehat{C}, \mathbf{x}, \{L^{(i)}_x, \overline{L}^{(i)}_x\}, \{L^{(i)}_y\}) :$
$\mathbf{w} = C_{\text{pub}}(\mathbf{x}),$
$\{L^{(j)}_w\}_{\mathbf{w}[j]=1} \leftarrow \mathsf{CDSEv}(C_{\text{pub}}, \mathsf{gb}, \mathbf{x}, \{L^{(i)}_x\}),$
$\{L^{(j)}_w\}_{\mathbf{w}[j]=0} \leftarrow \mathsf{CDSEv}(\overline{C_{\text{pub}}}, \mathsf{gb}, \mathbf{x}, \{\overline{L}^{(i)}_x\}),$
$\mathbf{z} \leftarrow \mathsf{BG.Eval}(C_{\text{priv}}, \widehat{C_{\text{priv}}}, \{L^{(j)}_w\}, \{L^{(i)}_y\}),$
Output $\mathbf{z}$.

1.4 From Partial to Full Garbling and PSM

We present a generic transformation that combines a succinct partial garbling scheme and an HE scheme for a program class $\mathcal{P}$ to obtain a succinct (fully hiding) garbling scheme for the same program class, which implies PSM for the same class. The transformation is extremely simple, and follows the blueprint of constructing succinct garbled circuits from FHE and ABE [GKP+13, BGG+14], replacing the ABE ingredient by partial garbling. To garble a program P, use the partial garbling scheme to garble the circuit $C(\hat{x}, \mathsf{sk}) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, P, \hat{x})) = P(x)$ that performs homomorphic evaluation of P over a public HE ciphertext $\hat{x}$ of the actual input x, followed by HE decryption using the secret key sk. The succinctness of partial garbling ensures that the garbled circuit $\hat{C}$ grows only with the complexity of HE decryption (i.e., $C_{\text{priv}}(\star, \star) = \mathsf{Dec}(\star, \star)$) which in turn depends only on the output length and the security parameter, if assuming circular security. Without circular security, the size additionally depends on the depth of the homomorphic evaluation of P (i.e., $C_{\text{pub}}(\star) = \mathsf{Eval}(\mathsf{pk}, P, \star)$). For the types of simple programs we consider here, the homomorphic evaluation depth is bounded by a fixed polynomial in the security parameter, eliminating the need for circular security.

We note that while using HE to ensure the privacy of the input x is a standard technique, our key observation here is that partial garbling suffices for ensuring the correctness / integrity of the homomorphic evaluation, replacing the stronger ABE techniques (with succinct secret keys) used in [GKP+13, BGG+14, QWW21]. This approach, though simple in retrospect, is crucial for moving away from lattice-based assumptions and iO.

Lemma 1 (From Partial to Full Garbling, Informal). *Any homomorphic encryption scheme for a class of programs $\mathcal{P}$ can be transformed into a succinct garbling scheme for $\mathcal{P}$, using a succinct partial garbling scheme able to support the homomorphic evaluation of the homomorphic encryption scheme for $\mathcal{P}$. The size of the garbled program is $\text{poly}(\lambda) \cdot m$ if assuming circular security, or $\text{poly}(\lambda) \cdot (m + D_{\mathsf{Eval}})$ without circular security, where m is the output length and D_{Eval} is the maximal circuit depth of the homomorphic evaluation of programs in $\mathcal{P}$.*

Succinct Garbling for Simple Programs and Implications for General Circuits. General-purpose FHE schemes are currently only known lattice assumptions or iO/FE, which this work tries to avoid. However, for several simple but useful classes of programs, there are

HE schemes relying on group-based assumptions. For instance, private information retrieval (PIR) schemes with polylogarithmic communication, which can be based on a variety of assumptions [KO97, Ste98, OS07, Lip05, GR05, DGI+19], can be viewed as a (compact) HE scheme for *truth-table* programs. This was generalized in [IP07, DGI+19] to yield, under similar assumptions, an HE scheme for *bounded-length branching programs* of unbounded size, where only the length bound impacts the encryption size.[5] As special cases, this enables the compact evaluation of decision trees and DFAs of an arbitrary size on an encrypted input. In a different direction, the Boneh-Goh-Nissim cryptosystem [BGN05] implies an HE scheme for quadratic polynomials over a finite field under an assumption on bilinear groups. Here the program size is at most quadratic in the input size.

Combining the above HE schemes with our succinct partial garbling, we obtain the following.

Corollary 1 (Succinct Garbling for Simple Programs). *Assuming DDH over prime-order groups, or Paillier/Damgård-Jurik groups, or class groups, there are succinct garbling schemes for the following classes:*

- *bounded-length (unbounded size) branching programs, assuming additionally the DDH assumption over prime order groups, or the DCR assumption over Paillier groups. This implies succinct garbling for truth tables, deterministic finite automata (DFAs), and decision trees of an arbitrary size.*
- *quadratic polynomials, assuming additionally the hardness of the subgroup decision problem in composite-order bilinear groups.*

We further show a generic composition theorem that leverages succinct garbling for simple programs, to garble circuits consisting of general gates computing these simple programs. The cost of garbling scales with the number of wires in circuits over general gates, without growing with the number of Boolean gates (such as AND gates) required to implement a general gate.

Corollary 2 (Implication for Garbling General Circuits). *Under the same assumptions as in Corollary 1, there is a garbling scheme for circuits over general gates, each computing one of the simple programs in Corollary 1, where the garbling size is* $\mathrm{poly}(\lambda) \cdot \#wires$*, where* $\#wires$ *is the number of wires in the circuit.*

Implication for PSM protocols. Our succinct garbling schemes imply the first group-based succinct PSM protocols for simple programs. In the case of truth tables, we get PSM protocols for arbitrary functions with polylogarithmic communication and polynomial computation in the input domain size.

Corollary 3 (Computationally Secure PSM for Truth Tables). *Assuming DDH over prime-order groups, or DCR plus DDH over Paillier groups, any* k*-party function* $f : [N]^k \to \{0,1\}$ *has a computationally secure PSM protocol with* $\mathrm{poly}(\lambda, k, \log N)$ *communication and* $\mathrm{poly}(\lambda, N^k)$ *computation.*

PSM vs. generalized secret sharing. Finally, it is interesting to compare our succinct computational PSM result for truth tables from Corollary 3 with a recent succinct computational secret sharing (SCSS) scheme from [ABI+23], which applies to *general* access structures represented by a truth table. One might a-priori expect the SCSS question to be easier because of

[5] This is analogous to LWE-based HE for circuits, where the ciphertext size depends on the circuit depth but not on its size.

its one-sided hiding requirement and the relation with partial garbling and CDS, for which we have strong positive results for truth tables even in the information-theoretic setting. However, the current state of the art on the two problems is incomparable. The group-based construction of SCSS from [ABI+23] specifically relies on the RSA assumption, and it is open whether the same conclusion holds from other group-based assumptions, such as the ones we use in this work. On the other hand, using LWE-based succinct garbling [BGG+14], succinct PSM can be based on LWE, which is still open for SCSS. The known group-based PSM and SCSS solutions are also incomparable in terms of the class of programs they efficiently support beyond truth tables: branching programs with bounded length in the PSM case and monotone CNF formulas of unbounded size in the SCSS case.

1.5 Overview of Constrained Pseudorandom Functions

By combining an aHMAC with HSS schemes, and following the HSS-based blueprint of Couteau, Meyer, Passelègue and Riahinia [CMPR23], we obtain the first CPRF for general constraint circuits from group-based assumptions. Note that the "bounded-depth" variant of aHMAC suffices for this application, and hence no circular security assumption is needed.

Theorem 3 (CPRFs for Circuits, Informal; see Theorem 13). *There is a 1-key, selectively-secure CPRF supporting general constraint circuits based on DCR, or DDH and small-exponent assumptions, over Paillier groups.*

Technical Overview In a constrained PRF (CPRF), there are two evaluation algorithms $\mathsf{Eval}, \mathsf{CEval}$, one with a normal key msk, and one with a constrained key sk_C with respect to a circuit C. Correctness requires evaluations (on $\mathbf{x}$) using both keys should equal if $C(\mathbf{x}) = 0$. Security requires evaluations using msk remain pseudorandom if $C(\mathbf{x}) = 1$, even given the constrained key sk_C.

$$\begin{array}{lll} & \mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda), \quad \mathsf{evk} \leftarrow \mathsf{Constrain}(\mathsf{msk}, C). & \\ \text{Correctness:} & \mathsf{Eval}(\mathsf{msk}, \mathbf{x}) = \mathsf{CEval}(\mathsf{sk}_C, \mathbf{x}), & \text{if } C(\mathbf{x}) = 0, \\ \text{Security:} & \mathsf{Eval}(\mathsf{msk}, \mathbf{x}) \text{ pseudorandom given } \mathsf{sk}_C & \text{o/w.} \end{array} \tag{3}$$

The Blueprint of [CMPR23]. The observation of [CMPR23] is that common homomorphic secret sharing schemes (HSS) for evaluating restricted multiplication straight-line programs (RMS) allows for an extended evaluation algorithm. Normally, an HSS evaluation for a function f locally transforms a pair of input shares I_0, I_1 for $\mathbf{x}$ into additive shares z_0, z_1 of $f(\mathbf{x})$:

$$\begin{aligned} I_0, I_1 &\leftarrow \mathsf{HSS.Input}(\mathbf{x}), \quad (\mathsf{evk}_0, \mathsf{evk}_1) \leftarrow \mathsf{HSS.Setup}(1^\lambda), \\ z_b &\leftarrow \mathsf{HSS.Eval}(b, \mathsf{evk}_b, I_b, f), \quad \text{s.t. } z_1 = z_0 + f(\mathbf{x}). \end{aligned}$$

The extended evaluation takes an additional pair of shares Δ_0, Δ_1 of the form $\Delta_1 = \Delta \cdot w + \Delta_0$ for some secret value Δ, and output additive shares z_0, z_1 of $w \cdot f(\mathbf{x})$. (See Section 5 for details of this extension.)

$$\begin{aligned} I_0, I_1 &\leftarrow \mathsf{HSS.Input}(\mathbf{x}), \quad (\mathsf{evk}_0, \mathsf{evk}_1, \Delta) \leftarrow \mathsf{HSS.Setup}(1^\lambda), \\ &\text{any } \Delta_0, \Delta_1 \text{ s.t. } \Delta_1 = \Delta_0 + \Delta \cdot w \\ z_b &\leftarrow \mathsf{HSS.ExtEval}(b, \mathsf{evk}_b, I_b, \Delta_b, f), \quad \text{s.t. } z_1 = z_0 + w \cdot f(\mathbf{x}). \end{aligned}$$

In order to construct a CPRF scheme, it now suffices to design a mechanism that let Eval and CEval respectively derive shares Δ_0, Δ_1 such that $\Delta_1 = \Delta \cdot C(\mathbf{x}) + \Delta_0$, and then run an extended HSS evaluation of the function $\mathsf{F}_{\mathbf{x}}(\cdot) = \mathsf{F}(\cdot, \mathbf{x})$, on input shares I_0, I_1 of a secret key key for a PRF F.

$$
\begin{aligned}
&\text{Blueprint:}\\
&\mathsf{msk} = (\mathsf{evk}_0, I_0, \dots,),\ \mathsf{sk}_C = (\mathsf{evk}_1, I_1, \dots,),\\
&\mathsf{Eval}: \quad \text{derive } \Delta_0, \quad z_0 \leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{evk}_0, I_0, \Delta_0, \mathsf{F}_{\mathbf{x}}),\\
&\mathsf{CEval}: \quad \text{derive } \Delta_1 \quad z_1 \leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{evk}_1, I_1, \Delta_1, \mathsf{F}_{\mathbf{x}}).
\end{aligned}
$$

By the extended evaluation correctness of HSS, we have $z_1 = z_0 + C(\mathbf{x}) \cdot \mathsf{F}(\mathsf{key}, \mathbf{x})$, which satisfy both CPRF correctness and security (Equation 3). The work of [CMPR23] uses another special HSS to let Eval, CEval derive the desired shares Δ_0, Δ_1. We instead use an aHMAC scheme for this.

In our construction of CPRF, we generate a secret key $\mathsf{aHMAC.sk}$ and evaluation key $\mathsf{aHMAC.evk}$ with respect to a user-supplied global secret Δ, which is exactly the secret for extended HSS evaluations. We view the constrained circuit C as a bit string, and compute tags $\{\sigma_C^{(i)}\}$ authenticating the bits of C.

$$
\begin{aligned}
\mathrm{KeyGen}(1^\lambda): &(\mathsf{evk}_0, \mathsf{evk}_1, \Delta) \leftarrow \mathsf{HSS.Setup}(1^\lambda),\\
&\mathsf{key} \leftarrow \$, \quad (I_0, I_1) \leftarrow \mathsf{HSS.Input}(\mathsf{key}),\\
&(\mathsf{aHMAC.sk}, \mathsf{aHMAC.evk}) \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \Delta).\\
&\text{Outputs } \mathsf{msk} = (I_0, \mathsf{evk}_0, I_1, \mathsf{evk}_1, \mathsf{aHMAC.sk}, \mathsf{aHMAC.evk}).\\
\mathrm{Constrain}(\mathsf{msk}, C): &\sigma_C^{(i)} \leftarrow \mathsf{aHMAC.Auth}(\mathsf{aHMAC.sk}, C[i], \mathsf{id}^{(i)}).\\
&\text{Outputs } \mathsf{sk}_C = (I_1, \mathsf{evk}_1, C, \{\sigma_C^{(i)}\}, \mathsf{aHMAC.evk}).
\end{aligned}
$$

Then, following the blueprint Eval and CEval can respectively run $\mathsf{aHMAC.EvalKey}$ and $\mathsf{aHMAC.EvalTag}$ with a universal function $U_{\mathbf{x}}(C) = C(\mathbf{x})$ to derive k_w and σ_w as the shares Δ_0 and Δ_1.

$$
\begin{aligned}
\mathrm{Eval}(\mathsf{msk}, \mathbf{x}): &\Delta_0 = k_U \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{aHMAC.sk}, U_{\mathbf{x}}, \{\mathsf{id}^{(i)}\}),\\
&z_0 \leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{evk}_0, I_0, \Delta_0, \mathsf{F}_{\mathbf{x}}),\\
\mathrm{CEval}(\mathsf{evk}, \mathbf{x}): &\Delta_1 = \sigma_U \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{aHMAC.evk}, U_{\mathbf{x}}, C, \{\sigma_C^{(i)}\}),\\
&z_1 \leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{evk}_1, I_1, \Delta_1, \mathsf{F}_{\mathbf{x}}).
\end{aligned}
$$

By the correctness of aHMAC, we indeed have $\Delta_1 = \Delta \cdot C(\mathbf{x}) + \Delta_0$.

1.6 Concurrent Work

In an independent and concurrent work, Liu, Wang, Yang, and Yu [LWYY24] constructed a garbling scheme for Boolean circuits that uses 1 bit per gate, based on RLWE or NTRU. There are several major differences between their work and ours. First, we achieve full succinctness, i.e., garbling size independent of the number of gates, whereas their garbled circuits still have linear size dependency. However, they achieve standard input privacy guarantees for general circuits, whereas our schemes either ensure only partial hiding, or only support simple classes of programs. Since succinct garbling was already known under LWE, a major focus of their work is improving concrete efficiency. In contrast, our schemes are based on group assumptions, diversifying the assumptions underlying succinct garbling.

1.7 Updates from the Previous Version [ILL24]

There are two main changes compared to the previous version of this paper:

- Whereas the previous version was centered around garbling, the title and introduction of the current version focus on the new homomorphic MAC primitive (aHMAC). The current version also includes more details on the connections between our aHMACs primitive and standard HMACs. See Section 3.5.
- The current version includes aHMAC constructions that avoid "powering-style" assumptions by applying a technique from [MORS25]. In particular, the previous leveled constructions from [ILL24] (in either the NIDLS framework or prime-order groups) assumed power-DDH (P-DDH) whereas the new constructions assume standard DDH. See Section 3.4.

For completeness, the current version also includes lattice-based constructions of aHMAC, which are powering-free variants of constructions from [ILL25]. See Appendix A.

2 Preliminaries

2.1 Garbling Schemes

In a garbling scheme, a program P and Boolean inputs $\mathbf{x}$ are encoded respectively into a garbled program $\widehat{P}$ and input labels $\{L_x^{(i)}\}$. The standard notion of garbling requires that $\widehat{P}, \{L_x^{(i)}\}$ together reveals nothing about the input $\mathbf{x}$ beyond $P(\mathbf{x})$. The notion of partial garbling, first proposed in [IW14], relaxes the security requirement to allow part of the input $\mathbf{x}$ to be leaked. We refer to this part as the public input.

The program can then be decomposed into two parts: $P(\mathbf{x}, \mathbf{y}) = P_{\text{priv}}(P_{\text{pub}}(\mathbf{x}), \mathbf{y})$, where P_{pub} represents the computation that depends on the public input $\mathbf{x}$ alone, and P_{priv} represents the rest that also depends on the private input $\mathbf{y}$. We call $P_{\text{pub}}, P_{\text{priv}}$ the public and private computations of P, respectively. We give two useful examples suitable for partial garbling:

- $P(\mathbf{x}, s) = f(\mathbf{x}) \cdot s$ implements a conditional disclosure of secret (CDS) functionality. The output of P reveals the secret input s if and only if the public input satisfy $f(\mathbf{x}) = 1$.
- $P(\mathsf{ct}, \mathsf{sk}) = \mathsf{HE.Dec}(\mathsf{HE.Eval}(f, \mathsf{ct}), \mathsf{sk})$ implements a homomorphic evaluation of HE ciphertexts ct, followed by decryption using a secret key sk. Assuming the ciphertexts correctly encrypts some input $\mathbf{x}$, then a partial garbling of P implements a (fully private) standard garbling of f.

In exchange for the relaxed security, we expect more efficient constructions. In this work, we consider the strong efficiency requirement, *succinctness with respect to public computation*, that the garbled program size $|\widehat{P}|$ should be independent of the complexity of the public computation. More precisely, we consider garbling families of programs $\mathcal{P} = \{\mathcal{P}_\lambda\}$ indexed by a security parameter λ. Each family $\mathcal{P}_\lambda$ satisfies some restrictions on its parameters, e.g. computation depth, but importantly has no polynomial bound on the *size* of the program. In this work, we obtain succinct partial garbling schemes for two classes:

- General Boolean circuits: $\mathcal{C} = \{\mathcal{C}_\lambda\}$, where $\mathcal{C}_\lambda$ consists of all two-input Boolean circuits, of form $C(\mathbf{x}, \mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}), \mathbf{y})$, based on circular-security assumptions, CP-DDH or KDM-DCR (Definition 9, 11);
- Bounded-depth Boolean circuits: $\mathcal{C}^d = \{\mathcal{C}_\lambda^d\}$, where $\mathcal{C}_\lambda^d$ consists of all two-input Boolean circuits with depth bounded by some fixed polynomial $d(\lambda)$, based on DDH (Definition 8).

As applications of our succinct partial garbling for bounded-depth circuits, we also obtain (fully private) succinct standard garbling schemes for two classes. Here succinctness requires the garbled program size $|\widehat{P}|$ to be independent of the complexity of the *entire* computation.

- Bounded-length branching programs: $\mathcal{P} = \{\mathcal{P}^\ell_\lambda\}$, where $\mathcal{P}^\ell_\lambda$ consists of branching programs with length bounded by some fixed polynomial $\ell(\lambda)$ and size below $2^{\mathrm{poly}(\lambda)}$.
- Quadratic polynomials (mod 2): $\mathcal{Q} = \{\mathcal{Q}_\lambda\}$, where $\mathcal{Q}_\lambda$ consists of quadratic polynomials with below $2^{\mathrm{poly}(\lambda)}$ number of monomials.

Besides succinctness, we also impose composability of garbled circuits at the syntax level: the garbled program evaluation should output arbitrary *target* key functions $K^{(i)}_z$ (specified at garbling time) applied to the output bits $\mathbf{z} = P(\mathbf{x}, \mathbf{y})$.

Definition 1 (Partial Garbling). *Let $\mathcal{P} = \{\mathcal{P}_\lambda\}$ be a class of programs with Boolean inputs and of the form $P(\mathbf{x}, \mathbf{y}) = P_{\mathrm{priv}}(P_{\mathrm{pub}}(\mathbf{x}), \mathbf{y})$, and $\mathcal{L} = \{\mathcal{L}_\lambda\}_\lambda$ be a label space of sizes $|\mathcal{L}_\lambda| \le 2^{\mathrm{poly}(\lambda)}$. A partial garbling scheme for $\mathcal{P}$ with label space $\mathcal{L}$ consists of two efficient algorithms:*

- $\mathsf{Garb}(1^\lambda, P \in \mathcal{P}_\lambda, \{K^{(i)}_z\}_{i\in[\ell_z]})$ *takes a program $P : \{0,1\}^{\ell_x} \times \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_z}$, and target key functions $\{K^{(i)}_z\}$ (mapping output bits to labels in $\mathcal{L}_\lambda$). It outputs a garbling $\widehat{P}$, and input key functions $\{K^{(i)}_x\}_{i\in[\ell_x]}$, and $\{K^{(i)}_y\}_{i\in[\ell_y]}$.*
- $\mathsf{Eval}(P, \widehat{P}, \{x^{(i)}, L^{(i)}_x\}_{i\in[\ell_x]}, \{L^{(i)}_y\}_{i\in[\ell_y]})$ *takes a program P, a garbling $\widehat{P}$, public inputs $x^{(i)}$, their labels $L^{(i)}_x$, and labels for private inputs $L^{(i)}_y$. It outputs labels $L^{(i)}_z$ for $i \in [\ell_z]$.*

Correctness: *For every polynomial $p(\lambda)$, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$, programs $P \in \mathcal{P}_\lambda$ with size $|P| \le p(\lambda)$, inputs $\mathbf{x}$, $\mathbf{y}$, and target key functions $\{K^{(i)}_z\}$ the following holds:*

$$\Pr\left[\begin{array}{l|r} \mathsf{Eval}(P, \widehat{P}, \{x^{(i)}, L^{(i)}_x\}, \{L^{(i)}_y\}) & (\widehat{P}, \{K^{(i)}_x\}, \{K^{(i)}_y\}) \leftarrow \mathsf{Garb}(1^\lambda, P, \{K^{(i)}_z\}), \\ = \{L^{(i)}_z\} & L^{(i)}_x = K^{(i)}_x(x^{(i)}),\ L^{(i)}_y = K^{(i)}_y(y^{(i)}), \\ & \mathbf{z} = P(\mathbf{x}, \mathbf{y}),\ L^{(i)}_z = K^{(i)}_y(z^{(i)}). \end{array}\right] \ge 1 - \mathrm{negl}(\lambda).$$

(Computational) Security: *There exists an efficient simulator* Sim *such that for every polynomial $p(\lambda)$, sequence of programs $\{P_\lambda\}$ from $\mathcal{P}$ such that $|P_\lambda| < p(\lambda)$, sequence of inputs $\{\mathbf{x}_\lambda, \mathbf{y}_\lambda\}$, and sequence of target key functions $\{K^{(i)}_{z,\lambda}\}_{i,\lambda}$, the following holds (suppressing the subscript λ for brevity):*

$$\left\{\mathsf{Sim}(1^\lambda, P, \mathbf{x}, \{L^{(i)}_z\}) \,|\, \mathbf{z} = P(\mathbf{x}, \mathbf{y}),\ L^{(i)}_z = K^{(i)}_z(z^{(i)})\right\}_\lambda$$
$$\approx_c \left\{\widehat{P}, \{L^{(i)}_x\}, \{L^{(i)}_y\} \;\middle|\; \begin{array}{l} (\widehat{P}, \{K^{(i)}_x\}, \{K^{(i)}_y\}) \leftarrow \mathsf{Garb}(1^\lambda, P), \\ L^{(i)}_x = K^{(i)}_x(x^{(i)}),\ L^{(i)}_y = K^{(i)}_y(y^{(i)}), \end{array}\right\}_\lambda$$

We now define the default succinctness requirement for partial garbling.

Definition 2 (Succinctness w.r.t. Public Computation). *We say a partial garbling scheme for a class $\mathcal{P}$ is* succinct w.r.t. public computation *if there exists a* $\mathrm{poly}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$ and $P \in \mathcal{P}_\lambda$, the garbling size $\widehat{P}$ is bounded by* $\mathrm{poly}(\lambda, |P_{\mathrm{priv}}|)$*, where $|P_{\mathrm{priv}}|$ denotes the complexity of the private computation P_{priv}.*

Note that a standard Boolean garbling scheme, e.g. Yao's garbling, can be viewed as a special case of our definition of partial garbling, where the public input is $\emptyset$. For standard garbling schemes, we define succinctness with respect to the entire (private) computation.

Definition 3 (Succinctness (w.r.t. Entire Computation)). *We say a (standard) garbling scheme for a class $\mathcal{P}$ is* succinct *if there exists a* $\mathrm{poly}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$ and $P \in \mathcal{P}_\lambda$, the garbling size $\widehat{P}$ is bounded by* $\mathrm{poly}(\lambda, \ell_z)$.

2.2 PSM and CDS

Definition 4 (Private Simultaneous Message (PSM) Schemes [FKN94, IK97]). *Let $\mathcal{P} = \{\mathcal{P}_\lambda\}$ be a class of programs with Boolean inputs and outputs. A PSM scheme for $\mathcal{P}$ consists of two efficient algorithms.*

- $\mathsf{Encode}(1^\lambda, P \in \mathcal{P}_\lambda, i \in [\ell_x], x \in \{0,1\}; \mathbf{r})$ *takes a program $P : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$, an input position i, an input bit x, and a shared randomness $\mathbf{r} \in \{0,1\}^\lambda$. It outputs a message msg_i.*
- $\mathsf{Recon}(1^\lambda, P, \{\mathsf{msg}_i\}_{[\ell_x]})$ *takes a program P and one message msg_i for each input position $i \in [\ell_x]$. It outputs an evaluation result $\mathbf{z} \in \{0,1\}^{\ell_z}$.*

Correctness. *For every polynomial $p(\lambda)$, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$, programs $P \in \mathcal{P}_\lambda$ with ℓ_x inputs and of size $|P| \le p(\lambda)$, and inputs $\mathbf{x} \in \{0,1\}^{\ell_x}$, the following holds:*

$$\Pr\left[\begin{array}{l|c} \mathsf{Recon}(1^\lambda, P, \{\mathsf{msg}_i\}) & \mathbf{r} \leftarrow \{0,1\}^\lambda, \text{ and } \forall i \in [\ell_x] \\ = P(\mathbf{x}) & \mathsf{msg}_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}[i]; \mathbf{r}) \end{array}\right] \ge 1 - \mathrm{negl}(\lambda).$$

(Computational) Privacy. *For every polynomial $p(\lambda)$, sequence of programs $\{P_\lambda\}$ from $\mathcal{P}$ such that $|P_\lambda| \le p(\lambda)$, sequences of inputs $\{\mathbf{x}_\lambda\}$, $\{\mathbf{x}'_\lambda\}$ satsifying $P_\lambda(\mathbf{x}_\lambda) = P_\lambda(\mathbf{x}'_\lambda)$, the following computational indistinguishability holds:*

$$\left\{\{\mathsf{msg}_i\} \;\middle|\; \begin{array}{c} \mathbf{r} \leftarrow \{0,1\}^\lambda, \text{ and } \forall i \in [\ell_x] \\ \mathsf{msg}_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}[i]; \mathbf{r}) \end{array}\right\}_\lambda \approx_c \left\{\{\mathsf{msg}'_i\} \;\middle|\; \begin{array}{c} \mathbf{r}' \leftarrow \{0,1\}^\lambda, \text{ and } \forall i \in [\ell_x] \\ \mathsf{msg}'_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}'[i]; \mathbf{r}') \end{array}\right\}_\lambda$$

Succinctness. *We say a PSM scheme for a class $\mathcal{P}$ is* succinct *if there exists a fixed* $\mathrm{poly}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$ and $P \in \mathcal{P}_\lambda$, the total message bit-lengths $\sum_i |\mathsf{msg}_i|$ is bounded by* $\mathrm{poly}(\lambda, \ell_z)$.

Remark 1. As outlined in [FKN94], a garbling scheme implements a multi-party private simultaneous messages (PSM) protocol as follows.

- Each party i running Encode computes a garbled circuit $\widehat{P}$ using shared randomness, and also a input label $L_x^{(i)}$. The message from this party consists of $\mathsf{msg}_i := (\widehat{P}, L_x^{(i)})$. As an optimization, we can also require only server 1 to include the garbled circuit $\widehat{P}$.
- The referee running Recon evaluates the garbled circuit $\widehat{P}$ with all input labels $\{L_x^{(i)}\}$ from each party to obtain the result $\mathbf{z} = P(\mathbf{x})$.

Therefore, a succinct standard garbling scheme for $\mathcal{P}$ directly implies a succinct PSM protocol.

Definition 5 (Disclosure of Secret (CDS) Schemes [GIKM00]). *Let $\mathcal{P} = \{\mathcal{P}_\lambda\}$ be a class of predicats with Boolean inputs and (1-bit) outputs. A CDS scheme for $\mathcal{P}$ consists of two efficient algorithms.*

- $\mathsf{Encode}(1^\lambda, P \in \mathcal{P}_\lambda, i \in [\ell_x], x \in \{0,1\}, \mathbf{z} \in \{0,1\}^{\ell_z}; \mathbf{r})$ *takes a predicate* $P : \{0,1\}^{\ell_x} \to \{0,1\}$*, an input position* i*, an input bit* x*, a shared secret* $\mathbf{z} \in \{0,1\}^{\ell_z}$*, and a shared randomness* $\mathbf{r} \in \{0,1\}^\lambda$*. It outputs a message* msg_i*.*
- $\mathsf{Recon}(1^\lambda, P, \mathbf{x} \in \{0,1\}^{\ell_x}, \{\mathsf{msg}_i\}_{[\ell_x]})$ *takes a predicate* P*, a public input* $\mathbf{x}$*, and one message* msg_i *for each input position* $i \in [\ell_x]$*. It outputs either* $\perp$ *or the secret* $\mathbf{z}$*.*

Correctness. *For every polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, predicates* $P \in \mathcal{P}_\lambda$ *with* ℓ_x *inputs and of size* $|P| \le p(\lambda)$*, inputs* $\mathbf{x} \in \{0,1\}^{\ell_x}$*, and secrets* $\mathbf{z} \in \{0,1\}^{\ell_z}$*, the following holds:*

$$\Pr\left[\begin{array}{l|c} \mathsf{Recon}(1^\lambda, P, \{\mathsf{msg}_i\}) & \mathbf{r} \leftarrow \{0,1\}^\lambda, \text{ and } \forall i \in [\ell_x] \\ = \mathbf{z} & \mathsf{msg}_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}[i], \mathbf{z}; \mathbf{r}) \end{array}\right] \ge 1 - \mathrm{negl}(\lambda).$$

(Computational) Privacy. *For every polynomial* $p(\lambda)$*, sequence of predicates* $\{P_\lambda\}$ *from* $\mathcal{P}$ *such that* $|P_\lambda| \le p(\lambda)$*, sequence of inputs* $\{\mathbf{x}_\lambda\}$*, satsifying* $P_\lambda(\mathbf{x}_\lambda) = 0$*, and sequence of secrets* $\{\mathbf{z}_\lambda\}$*, the following computational indistinguishability holds:*

$$\left\{\{\mathsf{msg}_i\} \;\middle|\; \begin{array}{c} \mathbf{r} \leftarrow \{0,1\}^\lambda, \text{ and } \forall i \in [\ell_x] \\ \mathsf{msg}_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}[i], \mathbf{z}; \mathbf{r}) \end{array}\right\}_\lambda \approx_c \left\{\{\mathsf{msg}'_i\} \;\middle|\; \begin{array}{c} \mathbf{r}' \leftarrow \{0,1\}^\lambda, \text{ and } \forall i \in [\ell_x] \\ \mathsf{msg}'_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}'[i], \mathbf{0}; \mathbf{r}') \end{array}\right\}_\lambda.$$

Succinctness. *We say a CDS scheme for a class* $\mathcal{P}$ *is* succint *if there exists a fixed* $\mathrm{poly}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*,* $P \in \mathcal{P}_\lambda$*, and secret* $\mathbf{z} \in \{0,1\}^{\ell_z}$*, the total message bit-lengths* $\sum_i |\mathsf{msg}_i|$ *is bounded by* $\mathrm{poly}(\lambda, \ell_z)$*.*

Remark 2. In the analogous way as using a standard garbling scheme to implement PSM, we can use a partial garbling scheme to implement CDS:

- Define $P'(\mathbf{x}, \mathbf{z}) := \mathbf{z} \cdot P(\mathbf{x})$.
- Each party i running Encode computes a garbled circuit $\widehat{P'}$ using shared randomness, an input label for its input $L_x^{(i)}$, and labels for the secret input $\{L_z^{(j)}\}$. The message from this party consists of $\mathsf{msg}_i := (\widehat{P'}, L_x^{(i)}, \{L_{\mathbf{z}}^{(j)}\})$. As an optimization, we can also require only server 1 to include the garbled circuit $\widehat{P'}$ and labels $\{L_z^{(j)}\}$ for the secret.
- The referee running Recon evaluates the garbled circuit $\widehat{P'}$ with the public input $\mathbf{x}$, corresponding public input labels $\{L_x^{(i)}\}$, and the secret input labels $\{L_z^{(j)}\}$ to obtain the result $\mathbf{z} \cdot P(\mathbf{x})$.

Therefore, a succinct partial garbling scheme for $\mathcal{P}$ directly implies a succinct CDS protocol.

2.3 Cryptographic Assumptions

In this work, we will use two types of groups: (1) groups satisfying the non-interactive distributed log sharing (NIDLS) framework [ADOS22], which have distributed discrete log (DDLog) algorithms with perfect correctness, and (2) prime-order groups, which have DDLog algorithms with a 1/poly correctness error [BGI16, DKK18].

Definition 6 (NIDLS Framework [ADOS22]). *Let* $\mathcal{G} = \{\mathcal{G}_\lambda\}$ *be a sequence of families of groups (with efficient group operations). We say* $\mathcal{G}$ *is an instantiation of the NIDLS framework if the following three efficient algorithms exist:*

– $\mathsf{Gen}(1^\lambda)$ *outputs public parameters* $\mathsf{pp} = (G, F, H, f, t, \ell)$ *where*
  - $G \in \mathcal{G}_\lambda$ *is a finite Abelian group with subgroups* F, H *s.t.* $G = F \times H$*;*
  - F *is a cyclic group of order* $t > 2^\lambda$*, and* f *is a generator of* F*;*
  - ℓ *is an upper-bound on the order of* H*.*
– $\mathsf{Samp}(\mathsf{pp})$ *samples an element* $g \in G$ *with the guarantee that* $f \in \langle g \rangle$*, and that the following statistical indistinguishability holds:*

$$\left\{ \mathsf{pp}, \rho, g^s \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s \leftarrow [\ell]. \end{array} \right\}$$
$$\approx \left\{ \mathsf{pp}, \rho, g' \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), g' \leftarrow \langle g \rangle. \end{array} \right\}.$$

  It outputs g *and the sampling randomness* ρ*.*
– $\mathsf{DDLog}(\mathsf{pp}, a \in G)$ *takes an element* a *and outputs a value* $\alpha \in \mathbb{Z}_t$ *with the guarantee that for all* $a \in G$*,* $m \in \mathbb{Z}_t$*:*

$$\mathsf{DDLog}(\mathsf{pp}, a \cdot f^m) = \mathsf{DDLog}(\mathsf{pp}, a) + m \bmod t.$$

Remark 3. Compared to the description in [ADOS22], we additionally require the subgroups F have large orders $t > 2^\lambda$. This is needed in our application to (non-interactively) convert additive shares mod t of $0, 1$ values into shares over $\mathbb{Z}$.

Known instantiations of the framework, with large subgroups F, include (the ciphertext spaces of) Damgård-Jurik encryption, a variant of Joye-Libert encryption described in [ADOS22], and class groups.

Definition 7 (Prime-order Groups). *We consider prime-order groups* $\mathcal{G} = \{\mathcal{G}_\lambda\}$ *that have efficient instance generation algorithms* Gen*:*

– $\mathsf{Gen}(1^\lambda)$ *outputs a group* $G \in \mathcal{G}_\lambda$ *with order* $p > 2^\lambda$*, and and a generator* g*. The group order* p *is included in the description of* G*.*

Lemma 2 (Distributed Discrete Log with Error [BGI16, DKK18]). *For any cyclic group* G *with order* p *and a generator* g*, there exists an algorithm* $\mathsf{DDLog}_{G,g}$*:*

– $\mathsf{DDLog}_{G,g}(\delta \in (0,1], B \in [p], \phi : G \to \{0,1\}^{\lceil \log(2B/\delta) \rceil}, a \in G)$ *takes an error bound* δ*, a message bound* B*, a function* ϕ *mapping group elements to bit strings, and an element* a*. It outputs a value* $\alpha \in \mathbb{Z}_p$*.*

The algorithm requires $O(\sqrt{B/\delta})$ *group operations, and has the guarantee that for all* $0 < \delta \leq 1$*,* $B < p$*,* $a \in G$*, and* $m \leq B$*:*

$$\Pr\left[\begin{array}{l} \mathsf{DDLog}_{G,g}(\delta, B, \phi, a \cdot g^m) \\ = \mathsf{DDLog}_{G,g}(\delta, B, \phi, a) + m \bmod p \end{array} \;\middle|\; \phi \leftarrow \$ \right] \geq 1 - \delta,$$

where $\phi \leftarrow \$$ *means sampling at random from all possible mappings.*

We state the standard DDH and power-DDH assumptions below, followed by our new circular-power-DDH and a proof that it holds in GGM.

Definition 8 (DDH Assumption). *We say the DDH assumption holds in the NIDLS framework if the following holds:*

$$\left\{ \mathsf{pp}, (\rho, g), g^a, g^b, g^{ab} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ a, b \leftarrow [\ell]. \end{array} \right\}$$
$$\approx_c \left\{ \mathsf{pp}, (\rho, g), g^a, g^b, g^c \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ a, b, c \leftarrow [\ell]. \end{array} \right\}.$$

We say the DDH assumption holds in prime-order groups if the following holds:

$$\left\{ G, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ a, b \leftarrow \mathbb{Z}_p. \end{array} \right\} \approx_c \left\{ G, g, g^a, g^b, g^c \;\middle|\; \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ a, b, c \leftarrow \mathbb{Z}_p. \end{array} \right\}.$$

Via a standard hybrid argument, we obtain DDH in matrix form:

Lemma 3 (DDH in Matrix Form). *Assuming DDH in the NIDLS framework, for any polynomials $m(\lambda)$, $n(\lambda)$, the following holds:*

$$\left\{ \mathsf{pp}, (\rho, g), g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{a} \cdot \mathbf{b}^T} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ \mathbf{a} \leftarrow [\ell]^m,\ \mathbf{b} \leftarrow [\ell]^n. \end{array} \right\}$$
$$\approx_c \left\{ \mathsf{pp}, (\rho, g), g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{C}} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), \mathbf{a} \leftarrow [\ell]^m, \mathbf{b} \leftarrow [\ell]^n, \mathbf{C} \leftarrow [\ell]^{m \times n}. \end{array} \right\}.$$

Similarly, assuming DDH in prime-order groups, for any polynomials $m(\lambda)$, $n(\lambda)$, the following holds:

$$\left\{ G, g, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{a} \cdot \mathbf{b}^T} \;\middle|\; \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathbf{a} \leftarrow \mathbb{Z}_p^m,\ \mathbf{b} \leftarrow \mathbb{Z}_p^n. \end{array} \right\}$$
$$\approx_c \left\{ G, g, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{C}} \;\middle|\; \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathbf{a} \leftarrow \mathbb{Z}_p^m,\ \mathbf{b} \leftarrow \mathbb{Z}_p^n, \mathbf{C} \leftarrow \mathbb{Z}_p^{m \times n}. \end{array} \right\}.$$

Definition 9 (Circular-Power-DDH Assumption). *We say the circular-power-DDH assumption holds in the NIDLS framework if the following holds:*

$$\left\{ \begin{array}{c} \mathsf{pp}, (\rho, g), g^s, g^{a_i}, g^{sa_i}, g^{s^2 a_i} \cdot f^{s[i]} \\ (\text{for } i \in \lceil \log \ell \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s, \{a_i\} \leftarrow [\ell]. \end{array} \right\}$$
$$\approx_c \left\{ \begin{array}{c} \mathsf{pp}, (\rho, g), g^s, g^{a_i}, g^{b_i}, g^{c_i} \\ (\text{for } i \in \lceil \log \ell \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s, \{a_i, b_i, c_i\} \leftarrow [\ell]. \end{array} \right\}.$$

We say the circular-power-DDH assumption holds in prime-order groups if the following holds:

$$\left\{ \begin{array}{c} G, g, g^s, g^{a_i}, g^{sa_i}, g^{s^2 a_i + s[i]} \\ (\text{for } i \in \lceil \log p \rceil) \end{array} \;\middle|\; \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ s, \{a_i\} \leftarrow \mathbb{Z}_p. \end{array} \right\}$$
$$\approx_c \left\{ \begin{array}{c} G, g, g^s, g^{a_i}, g^{b_i}, g^{c_i} \\ (\text{for } i \in \lceil \log p \rceil) \end{array} \;\middle|\; \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ s, \{a_i, b_i, c_i\} \leftarrow \mathbb{Z}_p. \end{array} \right\}.$$

Remark 4. CP-DDH implies DDH, which just requires indistinguishability of the first 3 terms in the above. We also show in Theorem 4 that CP-DDH in prime-order groups holds in the generic group model (GGM) as formulated in [Sho97].

The following small-exponent assumption is commonly assumed in the NIDLS framework, and is requires for obtaining HSS (for NC1 circuits) in prior work [ADOS22]. We don't rely on this assumption except when using existing HSS schemes as a black box.

Definition 10 (Small Exponent Assumption). *We say the small exponent assumption holds in the NIDLS framework if the following holds:*

$$\left\{ \mathsf{pp}, \rho, g^{s} \;\middle|\; \begin{array}{r} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s \leftarrow [\ell]. \end{array} \right\}$$
$$\approx_c \left\{ \mathsf{pp}, \rho, g^{s'} \;\middle|\; \begin{array}{r} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s' \leftarrow [2^\lambda]. \end{array} \right\}.$$

While the Damgård-Jurik encryption scheme [Pai99, DJ01] satisfy the NIDLS framework, our applications can alternatively rely directly on the KDM security of this scheme, rather than CP-DDH defined generically for the NIDLS framework. We give preliminaries for Damgård-Jurik below.

Construction 1 (Damgård-Jurik Encryption [Pai99, DJ01]). Let $B' = B'(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$ be a bound on message magnitude. The Damgård-Jurik encryption scheme for integer messages consists of the following algorithms.

- $\mathsf{KeyGen}(1^\lambda)$: sample two λ-bit primes p, q, set $N = p \cdot q$, and choose the smallest integer ζ such that $N^\zeta > B'$. Output $\mathsf{pk} = (N, \zeta)$ and $\mathsf{sk} = \varphi(N)$, where $\varphi(\cdot)$ is the Euler's totient function.
- $\mathsf{Enc}(\mathsf{pk}, m \in \mathbb{Z})$: sample $r \leftarrow \mathbb{Z}^*_{N^{\zeta+1}}$, and output a ciphertext

$$c = r^{N^\zeta} \cdot (1+N)^m \bmod N^{\zeta+1}.$$

- $\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c)$: compute and output

$$m = \mathsf{DLog}_{(1+N)}(c^{\mathsf{sk}})/\mathsf{sk} \bmod N^{\zeta+1},$$

  where $\mathsf{DLog}_{(1+N)}$ efficiently recovers x from $(1+N)^x \bmod N^{\zeta+1}$.

Definition 11 (KDM Security [BRS03, BHHO08]). *A public key encryption scheme is KDM secure w.r.t. a class of functions $\mathcal{F}$ if for every efficient adversary $\mathcal{A}$ there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$:*

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

where the experiment $\mathrm{Exp}^{\mathcal{A},\mathcal{F},b}_{\mathrm{KDM}}(\lambda)$ *is as follows:*

1. *Sample public and secret keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and launch* $\mathcal{A}(1^\lambda, \mathsf{pk})$.
2. *Answer arbitrary number of queries $f \in \mathcal{F}$ from $\mathcal{A}$ with*

$$c_f = \mathsf{Enc}(\mathsf{pk}, f(\mathsf{sk})) \qquad \text{if } b = 0$$
$$c_f = \mathsf{Enc}(\mathsf{pk}, 0^{|f(\mathsf{sk})|}) \qquad \text{if } b = 1.$$

3. In the end, $\mathcal{A}$ outputs a bit b' as the experiment result.

Remark 5. We will need the class $\mathcal{F}$ to contain constant functions $f_C(\mathsf{sk}) = C$, and inverse functions $f'_M(\mathsf{sk}) = \mathsf{sk}^{-1} \bmod M$ for all $C, M \in \mathbb{N}$.

Note that standard semantic security can be viewed as a special case of KDM security where $\mathcal{F}$ contains only constant functions. The semantic security of Damgård-Jurik encryption is also known as the DCR assumption. We write KDM-DCR as a shorthand for the KDM security of Damgård-Jurik encryption.

Lemma 4 (DDLog algorithm for Damgård-Jurik [RS21]). *Let p, q be any distinct primes, $\zeta \geq 1$ be any positive integer, $N = p \cdot q$, $\mathsf{pk} = (N, \zeta)$, and $\mathsf{sk} = \varphi(N)$. There exists an efficient algorithm $\mathsf{DDLog}_{N,\zeta}(\cdot)$, such that for all $x, y, z \in \mathbb{Z}$ and $c \in \mathrm{Supp}(\mathsf{Enc}(\mathsf{pk}, y))$, the following holds:*

$$\mathsf{DDLog}_{N,\zeta}(c^{\mathsf{sk} \cdot x + z}) \equiv \mathsf{sk} \cdot x \cdot y + \mathsf{DDLog}_{N,\zeta}(c^z) \mod N^\zeta.$$

2.4 CP-DDH in Generic Group Model

We first recall the definition of (Shoup's) generic group model [Sho97] as formulated in [Zha22], and then prove that our new assumption, CP-DDH in prime order groups, holds in this model (Theorem 4).

Definition 12 (GGM [Sho97, Zha22]). *Let $p \in \mathbb{Z}$ be a positive integer and $S \subseteq \{0,1\}^*$ be a set of strings of length bounded by some B, and cardinality at least p. In the generic group model for a cyclic group of order p, a random injective* labeling function $L : \mathbb{Z}_p \to S$ *is chosen, whose outputs $L(x)$ represents group elements g^x with respect to a fixed generator g. All parties – including the adversary and the challenger – are allowed the following queries (incurring unit cost) to a group oracle:*

- Labeling queries: *The party submits $x \in \mathbb{Z}_p$, and receives $L(x)$.*
- Group operations: *The party submits $(l_1, l_2, a_1, a_2) \in S^2 \times \mathbb{Z}_p^2$. If l_1, l_2 are valid labels for $x_1, x_2 \in \mathbb{Z}_p$, i.e. $L(x_1) = l_1, L(x_2) = l_2$, then the party receives the label $L(a_1 x_1 + a_2 x_2)$. Otherwise, the party receives $\perp$.*

Theorem 4 (CP-DDH in GGM). *For every sequence of prime orders $\{p_\lambda\}_\lambda$ where $p_\lambda > 2^\lambda$, and every adversary $\mathcal{A}$ with polynomial number of queries in GGM (for groups of orders $\{p_\lambda\}$), the following holds:*

$$|\Pr[\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{CPDDH}}) = 1] - \Pr[\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{Rand}}) = 1]| \leq \mathrm{negl}(\lambda),$$

where the labels provided to $\mathcal{A}$ are sampled as follows:

$$\mathsf{Labels}_{\lambda,\mathrm{CPDDH}} = \left\{ \begin{array}{l} L(1), L(s), L(a_i), L(sa_i), L(s^2 a_i + s[i]) \\ \quad (\text{for } i \in \lceil \log p_\lambda \rceil) \end{array} \middle| \; s, \{a_i\} \leftarrow \mathbb{Z}_{p_\lambda} \right\},$$

$$\mathsf{Labels}_{\lambda,\mathrm{Rand}} = \left\{ \begin{array}{l} L(1), L(s), L(a_i), L(b_i), L(c_i) \\ \quad (\text{for } i \in \lceil \log p_\lambda \rceil) \end{array} \middle| \; s, \{a_i, b_i, c_i\} \leftarrow \mathbb{Z}_{p_\lambda} \right\}.$$

Proof. We show a series of hybrid experiments whose output distribution transition from $\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{CPDDH}})$ to $\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{Rand}})$.

Hyb_0 : In this experiment, the challenger uniformly samples exponents $s, \{a_i\}$, queries the group oracle to obtain $\mathsf{Labels}_{\lambda,\mathrm{CPDDH}}$, and provides them to $\mathcal{A}$. The queries of $\mathcal{A}$ to the group oracle are answered by the oracle. The output of $\mathcal{A}$ is also the output of this experiment, i.e. $\mathsf{Hyb}_0 \equiv \mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{CPDDH}})$.

Hyb_1 : Instead of relying on the actual group oracle, the challenger simulates all its answers (to queries of both the challenger and the adversary) by lazily sampling a random label table L:

- Upon a labeling query of $x \in \mathbb{Z}_{p_\lambda}$: If it's not answered before, sample a random label $L(x) \leftarrow S$ and remember it. Otherwise, return $L(x)$.
- Upon a group operation query of (l_1, l_2, a_1, a_2): If either l_1 or l_2 is not in S, then return $\perp$. Otherwise, for an unqueried label, say l_1, randomly sample a previously un-queried $x_1 \in \mathbb{Z}_p$ and set $L(x_1) = l_1$. Finally, compute $x_3 = a_1x_1 + a_2x_2 \bmod p$ and return $L(x_3)$.

As the challenger perfectly simulates the group oracle, we have $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_0$.

Hyb_2 : Instead of simulating the group oracle as in Hyb_1, the challenger first translates every query into an affine function α over the values $s, \{a_i, sa_i, s^2a_i + s[i]\}$, and then answers the query as $L'(\alpha)$ with a random table L' mapping distinct translated affine functions to random labels in S.

The challenger's own labeling queries of exponents among $s, \{a_i, sa_i, s^2a_i+s[i]\}$ are translated to the affine functions that "select" the correct input variables. $\mathcal{A}$'s queries are handled as follows.

- Upon a labeling query of x: Translate it to the constant function $\alpha(\cdot) = x$.
- Upon a group operation query of (l_1, l_2, a_1, a_2): If either l_1 or l_2 is not in S, then return $\perp$. Otherwise, for an unqueried label, say l_1, randomly sample a previously un-queried constant function α_1, and set $L'(\alpha_1) = l_1$. Finally, translate the query to $\alpha_3 := a_1 \cdot \alpha_1 + a_2 \cdot \alpha_2$, which is another affine function.

We observe that the simulated answers in Hyb_2 and Hyb_1 are equivalent, unless *there exists queried affine functions $\alpha \neq \alpha'$ such that* $\alpha(s, \{a_i, sa_i, s^2a_i+s[i]\}) = \alpha'(s, \{a_i, sa_i, s^2a_i+s[i]\})$. We claim (and prove in the end) that this "bad event" happens with negligiable probability, because it implies a non-zero affine function $\alpha^* = \alpha - \alpha'$ satisfying $\alpha^*(s, \{a_i, sa_i, s^2a_i+s[i]\}) \equiv 0 \bmod p$:

Claim. For every prime p, every non-zero affine function α over $3\lceil \log p \rceil + 1$ inputs the following holds:

$$\Pr[\alpha(s, \{a_i, sa_i, s^2a_i + s[i]\}) \equiv 0 \bmod p \mid s, \{a_i\} \leftarrow \mathbb{Z}_p] \leq 3/p.$$

Therefore, we have $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$. We also note that in Hyb_2 the labels provided to $\mathcal{A}$, both as inputs $\mathsf{Labels}_{\lambda,\mathrm{CPDDH}}$ and as answers to its queries, are computed independent of the exponents $s, \{a_i\}$ sampled by the challenger.

Hyb_3 In this experiment, the challenger uniformly samples exponents $s, \{a_i, b_i, c_i\}$, queries the group oracle to obtain $\mathsf{Labels}_{\lambda,\mathrm{Rand}}$, and provides them to $\mathcal{A}$. The queries of $\mathcal{A}$ are answered by the oracle.

By exactly the same arguments as used to argue $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_2$ above, we have $\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{Rand}}) \equiv \mathsf{Hyb}_3 \approx \mathsf{Hyb}_2$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_3$, which proves the theorem. We prove the claim now.

Proof (of Claim). Denote the coefficients of α as $c, d, e_i, f_i, g_i \in \mathbb{Z}_p$ for $i \in \lceil \log p \rceil$:

$$\begin{aligned}
&\alpha(s, \{a_i, sa_i, s^2 a_i + s[i]\}) \\
:=&c + d \cdot s + \sum_i e_i \cdot a_i + \sum_i f_i \cdot sa_i + \sum_i g_i \cdot (s^2 a_i + s[i]) \\
=&\underbrace{c + sd + \sum_i s[i] g_i}_{\gamma} + \sum_i a_i \underbrace{(e_i + s f_i + s^2 g_i)}_{\beta_i}.
\end{aligned}$$

Let "Target" denote the event $\alpha(s, \{a_i, sa_i, s^2 a_i + s[i]\}) \equiv 0 \bmod p$. We analyze two possible cases:

Case A: $\gamma, \{\beta_i\}$ don't all evaluate to zero (mod p). By Schwartz-Zippel lemma, viewing $\{a_i\}$ as variables, we have

$$\Pr[\text{Target} \mid \text{Case A}] \leq 1/p$$

Case B: $\gamma, \{\beta_i\}$ all evaluate to zero (mod p). As we assume α is a non-zero function, at least one of the following are true:

- Exists i^* such $e_{i^*}, f_{i^*}, g_{i^*}$ are not all zero. By Schwartz-Zippel lemma, viewing s as the variable, we have $\Pr[\text{Case B}] \leq \Pr[\beta_{i^*} = 0] \leq 2/p$.
- $\{e_i, f_i, g_i\}$ are zero for all i, but c, d are not all zero. By Schwartz-Zipple lemma, viewing s as the variable, we have $\Pr[\text{Case B}] \leq \Pr[\gamma = 0] \leq 1/p$.

In both cases, we have

$$\Pr[\text{Case B}] \leq 2/p.$$

We conclude the proof by the following identity:

$$\begin{aligned}
\Pr[\text{Target}] =& \Pr[\text{Case A}] \Pr[\text{Target} \mid \text{Case A}] + \Pr[\text{Case B}] \Pr[\text{Target} \mid \text{Case B}] \\
\leq& 1 \cdot \Pr[\text{Target} \mid \text{Case A}] + \Pr[\text{Case B}] \cdot 1 \leq 3/p.
\end{aligned}$$

□

3 Algebraic Homomorphic MACs

Our notion of algebraic homomorphic MACs (aHMACs) can be roughly viewed as a refinement of existing homomorphic MACs (HMACs [AB09, GW13, CF13]). In both notions, there are Auth, and $\mathsf{EvalTag}$ algorithms that respectively produce authentication tags σ_x for some input x, and homomorphically evaluate some circuit C over them. The resulting tags σ_z should be verifiable with respect to the circuit C and unforgeable.

The main difference in our definition is the requirement that evaluated tags have the the form $\sigma_z = \Delta \cdot C(\mathbf{x}) + k_C$ (over $\mathbb{Z}$), where Δ is a global secret specified at key generation time, and k_C is computable from only the secret key and the circuit C, without knowing $\mathbf{x}$. This format is known as information-theoretic MACs. As our applications in this work doesn't require explicitely verifying the evaluated tags, we omit the Verify algorithm, verifiability, and unforgeability from our main Definition 13.

In Section 3.5, we provide the standard HMAC definition and show that our aHMAC definition (Definition 13) implies a HMAC scheme with verifiability and a weaker variant of unforgeability than commonly required in the literature. We also show how to generically amplify a scheme with weak unforgeability to achieve the usual unforgeability. (The amplified scheme doesn't satisfy the algebraic format anymore.)

Definition 13 (Algebraic Homomorphic MACs). *An algebraic homomorphic MAC scheme (for arbitrary Boolean circuits) consists of four efficient algorithms:*

- $\mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta} \in [2^\lambda]^{\ell_z})$ *takes a vector of global secrets* $\boldsymbol{\Delta}$*, (one for each evaluation output bit,) and outputs a secret key* sk *and evaluation key* evk*.*
- $\mathsf{Auth}(\mathsf{sk}, x \in \{0,1\}, \mathsf{id} \in \{0,1\}^\lambda)$ *takes as inputs the secret key* sk*, a bit* x *to be authenticated, and an* id *associated with the bit. It outputs a tag* σ_x*. We will write* $\sigma_{\mathbf{x}} = (\ldots, \sigma_x^{(i)}, \ldots)$ *to mean a vector of such tags.*
- $\mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$ *takes as inputs the evaluation key* evk*, a Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$*, input bits* $\mathbf{x}$*, and their associated tags* $\sigma_{\mathbf{x}}$*. It outputs tags* $\sigma_{\mathbf{z}} \in \mathbb{Z}^{\ell_z}$ *authenticating the outputs of* $C(\mathbf{x})$*.*
- $\mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$ *takes as inputs the secret key* sk*, a Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$ *and the* id*s associated with its inputs. It outputs MAC keys* $\mathbf{k}_C \in \mathbb{Z}^{\ell_z}$ *for the outputs of* C*.*

δ-Correctness: *Let* $\delta = \delta(\lambda)$ *be an error bound. For every polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, Boolean circuits* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$ *where* $|C| \le p(\lambda)$*, global secrets* $\boldsymbol{\Delta} \in [2^\lambda]^{\ell_z}$*, inputs* $\mathbf{x} \in \{0,1\}^{\ell_x}$*, and ids* $\mathbf{id} \in \{0,1\}^{\ell_x \times \lambda}$*, the following holds:*

$$\Pr\left[\begin{array}{l|l} \sigma_{\mathbf{z}} = \boldsymbol{\Delta} \odot C(\mathbf{x}) + \mathbf{k}_C & (\mathsf{sk},\mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta}) \\ (\text{over } \mathbb{Z}^{\ell_z}, \odot \text{ means} & \sigma^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \mathbf{id}[i]) \\ \text{component-wise mult}) & \sigma_{\mathbf{x}} := (\sigma^{(0)}, \ldots, \sigma^{(\ell_x - 1)}) \\ & \sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}) \\ & \mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id}) \end{array}\right] \ge 1 - \delta(\lambda) - \mathrm{negl}(\lambda).$$

(Adaptive) Security: *There exists a pair of efficient simulators* $\mathsf{Sim}_1, \mathsf{Sim}_2$ *such that for every efficient adversary* $\mathcal{A}$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, the following holds*

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

where the experiment $\mathrm{Exp}^{\mathcal{A},b}_{\mathrm{priv}}$ *is as follows:*

1. *Launch* $\mathcal{A}(1^\lambda)$*. Receive from* $\mathcal{A}$ *a vector* $\boldsymbol{\Delta} \in [2^\lambda]^{\ell_z}$*, compute* evk *as follows and send it to* $\mathcal{A}$*.*

$$\begin{cases} \mathsf{evk}, \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta}) & \text{if } b = 0, \\ \mathsf{evk}, \mathsf{st} \leftarrow \mathsf{Sim}_1(1^{\ell_z}) & \text{if } b = 1, \end{cases}$$

2. *Receive any number of adaptively chosen queries* $(x \in \{0,1\}, \mathsf{id} \in \{0,1\}^\lambda)$ *with distinct* id*s from* $\mathcal{A}$*, and answer each with a tag* σ *computed as follows.*

$$\begin{cases} \sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id}) & \text{if } b = 0, \\ \sigma, \mathsf{st} \leftarrow \mathsf{Sim}_2(\mathsf{st}) & \text{if } b = 1. \end{cases}$$

   (Queries with previously used id*s are ignored.)*
3. $\mathcal{A}$ *outputs a bit* b' *as the output of the experiment.*

Succinctness: *The tags produced by* Auth *and* $\mathsf{EvalTag}$ *have bounded sizes by some fixed* $\mathrm{poly}(\lambda)$*.*

Remark 6. The adaptive security definition implies the following weaker, but simpler, selective security that suffices for our applications. *There exists an efficient simulator* Sim *such that for every sequence of global MAC keys* $\{\mathbf{\Delta}_\lambda\}_\lambda$ *and inputs* $\{\mathbf{x}_\lambda\}_\lambda$ *(of polynomial lengths* $\ell_z(\lambda)$ *and* $\ell_x(\lambda)$*), and* distinct *ids* $\{\mathbf{id}_\lambda\}_\lambda$*, the following holds*

$$\left\{\mathsf{Sim}(1^\lambda, 1^{\ell_z})\right\}_\lambda \approx_c \left\{ \mathsf{evk}, \sigma_{\mathbf{x}} := (\sigma^{(i)}, \ldots, \sigma^{(\ell_x - 1)}) \;\middle|\; \begin{array}{l} (\mathsf{sk},\mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathbf{\Delta}) \\ \sigma^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \mathbf{id}[i]) \end{array} \right\}_\lambda .$$

We prove the stronger adaptive version for our constructions this section, but use the simpler version in our applications to succinct partial garbling (Section 4) and constrained PRF (Section 5).

Remark 7. We define a leveled variant analogously to Definition 13 with the following differences.

- KeyGen additionally takes a depth-bound D on evaluation circuits as 1^D.
- $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$ take circuits of depth less than D, and correctness only holds w.r.t. those circuits. (We don't require the circuits to be leveled.)
- Succinctness requires the bit-lengths of tags produced by Auth and $\mathsf{EvalTag}$ to be bounded by some fixed $\mathrm{poly}(\lambda, D)$.

In Section 3.1, 3.2, we show constructions of aHMACs assuming CP-DDH in the NIDLS framework or in prime-order groups. In Section 3.3, we describe a construction assuming KDM-DCR (in Damgård-Jurik groups). In Section 3.4, we describe leveled variants of these constructions assuming DDH in the NIDLS framework or in prime-order groups. (A leveled variant assuming DCR is described in a recent concurrent work [MORS25], we only include this in our theorem statement for completeness, but don't claim any credit. Our leveled constructions assuming DDH are also inspired by the techniques from [MORS25].) In Section 3.5 we show that our aHMAC definition implies a HMAC scheme with verifiability and (a weaker variant of) unforgeability. We also show how to generically amplify weak unforgeability to standard unforgeability. Finally, in Appendix A we include a construction based on lattice assumptions from subsequent work [ILL25], and a leveled variant analogous to our group-based constructions for completeness.

Our results on aHMACs are summarized as follows. (See Theorem 7, 8 in Section 3.5 for the implied results on HMACs.)

Theorem 5 (aHMACs). *We have the following constructions:*

1. *Assuming CP-DDH in the NIDLS framework (e.g. Damgård-Jurik groups and class groups) (Definition 6, 9), or KDM-DCR (Definition 11), there exists an aHMAC scheme achieving* negl*-correctness.*
2. *Assuming CP-DDH in prime-order groups, for every polynomial* $p(\lambda)$*, there exists an aHMAC scheme achieving* $1/p$*-correctness.*

In the above, evk *costs* $\ell_z \cdot \mathrm{poly}(\lambda)$ *bits.*

Theorem 6 (Leveled aHMACs). *We have the following constructions (besides those from Theorem 5):*

1. *Assuming DDH in the NIDLS framework (Definition 8), or DCR,* [6] *there exists a leveled aHMAC scheme achieving* negl*-correctness.*

[6] As noted, the leveled contruction under DCR is adapted from [MORS25].

2. *Assuming DDH in prime-order groups, for every polynomial $p(\lambda)$, there exists a leveled aHMAC scheme achieving $1/p$-correctness.*

In the above, evk *costs* $(\ell_z + D) \cdot \mathrm{poly}(\lambda)$ *bits.*

3.1 aHMACs from the NIDLS Framework

In this section, we construct an aHMAC scheme in the NIDLS framework (Definition 6). See Section 1.2 for an overview and intuitions.

Construction 2 (aHMACs from the NIDLS Framework). Ingredients:

- An instance $\mathcal{G} = \{\mathcal{G}_\lambda\}$ of the NIDLS framework with large order F, i.e., the subgroup F of each $G \in \mathcal{G}_\lambda$ has order at least $t > 2^\lambda$.
- Two PRFs $\mathsf{F}_1 : \mathcal{K}_1 \times \{0,1\}^* \to [t]$ and $\mathsf{F}_2 : \mathcal{K}_2 \times \{0,1\}^* \to [2^\lambda]$.

Note that every Boolean circuit C can be implemented via an arithmetic circuit C' over $\mathbb{Z}$ as follows:

$$\forall x, y \in \{0,1\}, \quad x \,\mathrm{AND}\, y = x \cdot y, \quad x \,\mathrm{OR}\, y = x + y - x \cdot y, \quad \mathrm{Not}\, x = 1 - x.$$

The wire values in C' are 0 or 1. In the following construction of $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$, we will evaluate C' instead of C.

For an integer vector $\mathbf{r}$, we write $g^{\mathbf{r}} = (\ldots, g^{\mathbf{r}[i]}, \ldots)$, denote component-wise multiplication by $\odot$, and define $\mathsf{BC}(\mathbf{r}) = \sum_i \mathbf{r}[i] \cdot 2^i$ over $\mathbb{Z}$. When using a PRF to generate n values from a single input, we write $\mathsf{F}^n(s, x) = (\ldots, \mathsf{F}(s, x\|i), \ldots)_{i \in [n]}$ for vectorized operations.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathbf{\Delta})$:
Generate public parameters $\mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda)$, a group element $(g, \rho) \leftarrow \mathsf{Samp}(\mathsf{pp})$, and a secret exponent $s \leftarrow [\ell]$. Then compute ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ encrypting the bits of s (as a bit vector $\bar{\mathbf{s}} := \mathsf{Bits}(s)$ of length $\ell_s := \lceil \log \ell \rceil$, such that $\mathsf{BC}(\bar{\mathbf{s}}) = s$) and of $\mathbf{\Delta}$ (as a bit matrix in $\{0,1\}^{\ell_z \times \lambda}$):

$$\begin{aligned} \mathbf{h} = g^{\mathbf{r}}, \mathbf{r} \leftarrow [\ell]^{\ell_s}, \qquad & \mathbf{H} = g^{\mathbf{R}}, \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, \\ \mathsf{ct}_s = (\mathbf{h}, \mathbf{h}^s, \mathbf{h}^{s^2} \cdot f^{\bar{\mathbf{s}}}), \quad & \mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}^{-s} \cdot f^{\mathbf{\Delta}}). \end{aligned} \tag{4}$$

Finally, sample PRF keys $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, $\mathsf{key}_2 \leftarrow \mathcal{K}_2$. Output $\mathsf{sk} = (\mathsf{pp}, \mathbf{h}, \mathbf{H}, \bar{\mathbf{s}}, \mathsf{key}_1, \mathsf{key}_2)$, and $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$.

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$:
Parse the secret exponent $\bar{\mathbf{s}}$ and the (secret) PRF key key_2 from sk. Then compute an authentication tag

$$\sigma_x = \bar{\mathbf{s}} \cdot x + \mathsf{F}_2^{\ell_s}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathbb{Z}^{\ell_s}.$$

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$:
Parse pp, ciphertexts $\mathsf{ct}_s = \{\mathbf{h}, \mathbf{h}_1, \mathbf{h}_2\}, \mathsf{ct}_\Delta = \{\mathbf{H}, \mathbf{H}_1\}$, and a PRF key key_1 from evk.

1. Assign the tags $\sigma_{\mathbf{x}}$ to corresponding input wires of C', and then a tag $\sigma^{(w)}$ to every output wire w of some gate in C' (with input wires w_1, w_2 and values x_1, x_2) following the topological order:
   - For Add gates, set $\sigma^{(w)} := \sigma^{(w_1)} + \sigma^{(w_2)}$ over $\mathbb{Z}^{\ell_s}$.

– For Mult gates, compute the output tag $\sigma^{(w)}$ as follows:

$$\mathbf{a}^{(w)} := \mathbf{h}^{\mathsf{BC}(\sigma^{(w_1)})\cdot\mathsf{BC}(\sigma^{(w_2)})} \odot \mathbf{h}_1^{-\mathsf{BC}(\sigma^{(w_1)})\cdot x_2 - \mathsf{BC}(\sigma^{(w_2)})\cdot x_1} \odot \mathbf{h}_2^{x_1\cdot x_2},$$
$$\sigma^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{a}^{(w)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \mod t.$$

We note the following invariant: if the input tags have the form $\sigma^{(w_1)} = \overline{\mathbf{s}}\cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \overline{\mathbf{s}}\cdot x_2 + \mathbf{k}^{(w_2)}$, over $\mathbb{Z}$, then the computed tag also has the form $\sigma^{(w)} = \overline{\mathbf{s}}\cdot z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$. For Add gates, the invariant is immediate, with $\mathbf{k}^{(w)} := \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}$. For Mult gates, we note the following core identity:

$$\begin{aligned}&\mathsf{BC}(\sigma^{(w_1)})\mathsf{BC}(\sigma^{(w_2)}) - s\cdot\Big(\mathsf{BC}(\sigma^{(w_1)})x_2 + \mathsf{BC}(\sigma^{(w_2)})x_1\Big) + s^2 z\\ =&\mathsf{BC}(\mathbf{k}^{(w_1)})\mathsf{BC}(\mathbf{k}^{(w_2)}) \quad \text{over } \mathbb{Z}.\end{aligned}$$

Plugging in the fact that $\mathbf{h}_1 = \mathbf{h}^s$, $\mathbf{h}_2 = \mathbf{h}^{s^2}\cdot f^{\overline{\mathbf{s}}}$, we obtain

$$\begin{aligned}&\mathbf{a}^{(w)} = f^{\overline{\mathbf{s}}\cdot z}\cdot \mathbf{h}^{\mathsf{BC}(\mathbf{k}^{(w_1)})\mathsf{BC}(\mathbf{k}^{w_2})}\\ \Rightarrow&\mathsf{DDLog}(\mathsf{pp}, \mathbf{a}^{(w)}) = \overline{\mathbf{s}}\cdot z + \mathsf{DDLog}(\mathsf{pp}, \mathbf{h}^{\mathsf{BC}(\mathbf{k}^{(w_1)})\mathsf{BC}(\mathbf{k}^{(w_2)})}) \bmod t.\\ \Rightarrow&\sigma^{(w)} = \overline{\mathbf{s}}\cdot z + \mathbf{k}^{(w)} \bmod t,\\ &\text{w/ } \mathbf{k}^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{h}^{\mathsf{BC}(\mathbf{k}^{(w_1)})\mathsf{BC}(\mathbf{k}^{(w_2)})}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \bmod t.\end{aligned}$$

We have obtained the desired invariant mod t, and now argue it also holds over $\mathbb{Z}$. For each coordinate i, there are at most $\|\overline{\mathbf{s}}\cdot z\|_\infty \le 1$ possible values of $\mathbf{k}^{(w)}[i]$ to break the invariant over $\mathbb{Z}$. Since $\mathbf{k}^{(w)}$ is distributed pseudorandomly mod t, due to the offset by F_1, the probability of it breaking the invariant is $\le (1/t)^{\ell_s} = \mathrm{negl}(\lambda)$.

2. Compute the final output tags $\sigma_{\mathbf{z}} = (\ldots, \mathsf{BC}(\sigma'^{(o_j)}), \ldots)_{j\in[\ell_z]}$, where $\{o_j\}$ are the output wires of C' (with values $\{z_j\}$):

$$\mathbf{a}'^{(o_j)} := \mathbf{H}[j]^{\mathsf{BC}(\sigma^{(o_j)})} \odot \mathbf{H}_1[j]^{z_j},$$
$$\sigma'^{(o_j)} = \mathsf{DDLog}(\mathsf{pp}, \mathbf{a}'^{(o_j)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, o_j) \mod t.$$

Similarly, we note if the tags $\sigma^{(o_j)}$ have the form $\sigma^{(o_j)} = \overline{\mathbf{s}}\cdot z_j + \mathbf{k}^{(o_j)}$, then we have

$$\begin{aligned}&\sigma'^{(o_j)} = \boldsymbol{\Delta}[j]\cdot z_j + \mathbf{k}'^{(o_j)} \quad \text{over } \mathbb{Z},\\ &\text{w/ } \mathbf{k}'^{(o_j)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{H}[j]^{\mathsf{BC}(\mathbf{k}^{(o_j)})}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, o_j) \bmod t.\\ \Rightarrow&\sigma_{\mathbf{z}} = \boldsymbol{\Delta}\odot z_j + \mathbf{k}_C \quad \text{over } \mathbb{Z} \quad \text{w/ } \mathbf{k}_C := \mathsf{BC}(\mathbf{k}'^{(o_j)}),\end{aligned}$$

where in the last line we abuse notations to write $\boldsymbol{\Delta}$ as a vector in $\mathbb{Z}^{\ell_z}$.

$\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$:
Parse the PRF keys $\mathsf{key}_1, \mathsf{key}_2$ and group elements $\mathbf{h}, \mathbf{H}$ from sk. Then compute MAC keys $\mathbf{k}^{(w_j)}$ associated with each input wire w_j of C':

$$\mathbf{k}^{(w_j)} = \mathsf{F}_2^{\ell_s}(\mathsf{key}_2, \mathbf{id}[j]).$$

1. Assign a MAC key to every output wire w of some gate in C' (with input wires w_1, w_2) following the topological order:
   - For Add gates, set $\mathbf{k}^{(w)} := \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}^{\ell_s}$.
   - For Mult gates, compute the output MAC key $\mathbf{k}^{(w)}$ as follows:

$$\begin{aligned}\mathbf{b}^{(w)} &:= \mathbf{h}^{\mathsf{BC}(\mathbf{k}^{(w_1)})\cdot\mathsf{BC}(\mathbf{k}^{(w_2)})},\\ \mathbf{k}^{(w)} &:= \mathsf{DDLog}(\mathsf{pp}, \mathbf{b}^{(w)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \mod t.\end{aligned}$$

   As noted before, we have $\sigma^{(w)} = \overline{\mathbf{s}} \cdot z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$.

2. Compute the final output MAC keys $\mathbf{k}_C = (\ldots, \mathsf{BC}(\mathbf{k}'^{(o_j)}), \ldots)_{j\in\ell_z}$, where $\{o_j\}$ are the output wires of C':

$$\begin{aligned}\mathbf{b}'^{(o_j)} &:= \mathbf{H}[j]^{\mathsf{BC}(\mathbf{k}^{(o_j)})},\\ \mathbf{k}'^{(o_j)} &= \mathsf{DDLog}(\mathsf{pp}, \mathbf{b}'^{(o_j)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, o_j) \mod t.\end{aligned}$$

   As noted before, we have $\sigma_{\mathbf{z}} = \boldsymbol{\Delta} \odot \mathbf{z} + \mathbf{k}_C$ over $\mathbb{Z}$ as desired.

Correctness and Efficiency: To help digest the construction, we have broken up and embedded correctness analysis as notes in the above. We note that the tags output by Auth and the $\mathsf{EvalTag}$ all have bounded magnitude by $O(t) \le O(2^\lambda)$. Hence they have bit-lengths bounded by $O(\lambda \cdot \ell_s) = \mathrm{poly}(\lambda)$, and satisfy succinctness. The evaluation key evk contains mainly the ciphertexts ct_s, ct_Δ, which are $O(\ell_s + \ell_z \times \lambda)$ group elements. In total, evk has bit-length $\ell_z \cdot \mathrm{poly}(\lambda)$.

Security: We state and prove the following security lemma.

Lemma 5. *Under CP-DDH in the NIDLS framework, Construction 2 is secure.*

Proof (of Lemma 7). The security of an aHMAC scheme (Definition 13) requires simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ to simulate an evaluation key evk and adaptively queried authentication tags σ.

- Sim_1 samples all components of the simulated $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$ at random. In more detail, it samples a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, public parameters of a NIDLS group $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, and a random group element $(g, \rho) \leftarrow \mathsf{Samp}(\mathsf{pp})$. It then samples random ciphertexts $\mathsf{ct}_s = (\widetilde{\mathbf{h}}, \widetilde{\mathbf{h}}_1, \widetilde{\mathbf{h}}_2)$, and $\mathsf{ct}_\Delta = (\widetilde{\mathbf{H}}, \widetilde{\mathbf{H}}_1)$:

$$\begin{aligned}&\widetilde{\mathbf{h}} = g^{\mathbf{r}}, \mathbf{r} \leftarrow [\ell]^{\ell_s}, \quad \widetilde{\mathbf{h}}_1 = g^{\mathbf{r}_1}, \mathbf{r}_1 \leftarrow [\ell]^{\ell_s}, \quad \widetilde{\mathbf{h}}_2 = g^{\mathbf{r}_2}, \mathbf{r}_2 \leftarrow [\ell]^{\ell_s},\\ &\widetilde{\mathbf{H}} = g^{\mathbf{R}}, \mathbf{R} \leftarrow [\ell]^{\ell_z\times\lambda}, \quad \widetilde{\mathbf{H}}_1 = g^{\mathbf{R}_1}, \mathbf{R}_1 \leftarrow [\ell]^{\ell_z\times\lambda}.\end{aligned}$$

- Sim_2 samples the authentication tag at random $\widetilde{\sigma} \leftarrow [2^\lambda]^{\ell_s}$.

We show a series of hybrids that transitions from the real-world experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^0_{\mathrm{priv}}$ in Definition 13 to the simulation-world experiment $\mathsf{Hyb}_5 = \mathrm{Exp}^1_{\mathrm{priv}}$.

Hyb_0 : We summarize the real-world distribution of the evaluation key $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}_s = (\mathbf{h}, \mathbf{h}_1, \mathbf{h}_2)$, and $\mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}_1)$, and of the authentication tag σ for some query (x, id).

$$\begin{aligned} &\mathsf{key}_1 \leftarrow \mathcal{K}_1, \quad \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda), \\ &\mathbf{h} = g^{\mathbf{r}},\ \mathbf{h}_1 = g^{s\cdot\mathbf{r}},\ \mathbf{h}_2 = g^{s^2\cdot\mathbf{r}} \cdot f^{\overline{\mathbf{s}}}, && \left|\begin{array}{l}(\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), \mathbf{r} \leftarrow [\ell]^{\ell_s}, \\ \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, s \leftarrow [\ell].\end{array}\right. \quad (5) \\ &\mathbf{H} = g^{\mathbf{R}},\ \mathbf{H}_1 = g^{s\cdot\mathbf{R}} \cdot f^{\mathbf{\Delta}}, && \\ &\sigma = \overline{\mathbf{s}} \cdot x + \mathsf{F}_2^{\ell_s}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathbb{Z} && \left|\mathsf{key}_2 \leftarrow \mathcal{K}_2, \overline{\mathbf{s}} := \mathsf{Bits}(s).\right. \quad (6) \end{aligned}$$

Hyb_1 : Instead of computing each tag σ as in Equation 6, Hyb_1 simulates it as $\widetilde{\sigma} \leftarrow [2^\lambda]^{\ell_s}$. The PRF security of F_2 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Instead of sampling the random exponents $\mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}$ as in Equation 5, Hyb_2 simulate it as $\widetilde{\mathbf{R}} = \mathbf{r}' \cdot \mathbf{r}^T$, where $\mathbf{r}' \leftarrow [\ell]^{\ell_z}$. [7] The matrix form of DDH (Lemma 3) in the NIDLS framework ensures that $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_1$.

To summarize, in Hyb_2 the terms $\mathbf{h}$, $\mathbf{h}_1$, $\mathbf{h}_2$, $\mathbf{H}$, $\mathbf{H}_1$ are computed as:

$$\begin{aligned} &\mathbf{h} = g^{\mathbf{r}},\ \mathbf{h}_1 = g^{s\cdot\mathbf{r}},\ \mathbf{h}_2 = g^{s^2\cdot\mathbf{r}} \cdot f^{\overline{\mathbf{s}}}, && \left|\begin{array}{l}(\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), \mathbf{r} \leftarrow [\ell]^{\ell_s}, \\ \mathbf{r}' \leftarrow [\ell]^{\ell_z}, s \leftarrow [\ell].\end{array}\right. \\ &\mathbf{H} = g^{\mathbf{r}'\cdot\mathbf{r}^T},\ \mathbf{H}_1 = g^{\mathbf{r}'\cdot(s\cdot\mathbf{r})^T} \cdot f^{\mathbf{\Delta}}, && \end{aligned}$$

In particular, $\mathbf{H}$ and $\mathbf{H}_1$ can be derived from $\mathbf{h}$, $\mathbf{h}_1$, $\mathbf{r}'$ and $\mathbf{\Delta}$.

Hyb_3 : Instead of computing $\mathbf{h}$, $\mathbf{h}_1$, $\mathbf{h}_2$ as above, Hyb_3 simulates:

$$\widetilde{\mathbf{h}} = g^{\mathbf{a}}, \quad \widetilde{\mathbf{h}}_1 = g^{\mathbf{b}}, \quad \widetilde{\mathbf{h}}_2 = g^{\mathbf{c}}, \,|\, (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow [\ell]^{\ell_s}.$$

CP-DDH in the NIDLS framework ensures that $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

In Hyb_3, the terms $\mathbf{H}$, $\mathbf{H}_1$ (derived from $\widetilde{\mathbf{h}}$ and $\widetilde{\mathbf{h}}_1$) becomes

$$\mathbf{H} = g^{\mathbf{r}'\cdot\mathbf{a}^T},\ \mathbf{H}_1 = g^{\mathbf{r}'\cdot\mathbf{b}^T} \cdot f^{\mathbf{\Delta}}, \,|\, \mathbf{r}' \leftarrow [\ell]^{\ell_z}.$$

Hyb_4 : Instead of computing $\mathbf{H}$, $\mathbf{H}_1$ as above, Hyb_4 simulates them as

$$\widetilde{\mathbf{H}} = g^{\mathbf{R}},\ \widetilde{\mathbf{H}}_1 = g^{\mathbf{R}_1} \cdot f^{\mathbf{\Delta}} \,|\, \mathbf{R}, \mathbf{R}_1 \leftarrow [\ell]^{\ell_z \times \lambda}.$$

The matrix form of DDH in the NIDLS framework ensures $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_3$.

Hyb_5 : Instead of computing $\widetilde{\mathbf{H}}_1$ as above, Hyb_5 simulates $\widetilde{\mathbf{H}}_1 = g^{\mathbf{R}_1}$, i.e., independent of $\mathbf{\Delta}$. The Samp algorithm (Definition 6) ensures

$$g^{\mathbf{R}_1} \cdot f^{\mathbf{\Delta}} \approx \mathsf{Uniform}(\langle g \rangle) \cdot f^{\mathbf{\Delta}} \equiv \mathsf{Uniform}(\langle g \rangle) \approx g^{\mathbf{R}_1},$$

where the first and last indistinguishabilities are statistical. Hence we have $\mathsf{Hyb}_5 \approx \mathsf{Hyb}_4$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_5$, which proves the lemma. □

[7] An omitted detail (for brevity) here is the mismatch of dimensions: $\widetilde{\mathbf{R}}$ should have dimension $\ell_z \times \lambda$, but $\mathbf{r}' \cdot \mathbf{r}^T$ has dimension $\ell_z \times \ell_s$, where $\ell_s = \lceil \log \ell \rceil \geq \lambda$. We simply take $\mathbf{R}$ to be the first λ columns of $\mathbf{r}' \cdot \mathbf{r}^T$.

3.2 aHMACs from Prime-Order Groups

In this section, we construct an aHMAC scheme in prime-order groups. It follows the same blue-print as Construction 2 in the NIDLS framework, but will have a 1/poly-correctness error due to the imperfect DDLog algorithm in prime-order groups.

Construction 3 (aHMACs from Prime-Order Groups). Ingredients:

- Prime-order groups $\mathcal{G} = \{\mathcal{G}_\lambda\}$ with an algorithm Gen and orders $p > 2^\lambda$.
- A compatible PRF $\mathsf{F}_3 : \mathcal{K}_3 \times G \to \{0,1\}^\lambda$ used for the DDLog algorithm.
- Two PRFs $\mathsf{F}_1 : \mathcal{K}_1 \times \{0,1\}^* \to [p]$ and $\mathsf{F}_2 : \mathcal{K}_2 \times \{0,1\}^* \to [2^\lambda]$.

Compared to Construction 2, the Auth, $\mathsf{EvalTag}$, $\mathsf{EvalKey}$ algorithms are the same except DDLog now requires three additional parameters δ', B, ϕ (Lemma 2). We set $\delta' = \delta/O(|C|)$ so that running DDLog $O(|C|)$ times has an overall error probability bounded by δ, and $B = 1$ which equals the wire value bound when implementing C as an arithmetic circuit (as explained in Construction 2). We sample a public PRF key key_3 during KeyGen which specifies the function $\phi(\cdot) := \mathsf{F}_3(\mathsf{key}_3, \cdot)$. [8] The remaining differences are in KeyGen, where we compute $\mathsf{ct}_s, \mathsf{ct}_\Delta$ as part of evk differently:

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \Delta)$:

Generate public parameters $\mathsf{pp} = (G, g) \leftarrow \mathsf{Gen}(1^\lambda)$ and a secret exponent $s \leftarrow \mathbb{Z}_p$. Then compute ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ encrypting the bits of s (as a bit vector $\overline{\mathbf{s}} \in \{0,1\}^{\ell_s}$) and $\mathbf{\Delta}$ (as a bit matrix in $\{0,1\}^{\ell_z \times \lambda}$):

$$\mathbf{h} = g^{\mathbf{r}}, \mathbf{r} \leftarrow \mathbb{Z}_p^{\ell_s}, \qquad \mathbf{H}, = g^{\mathbf{R}}, \mathbf{R} \leftarrow \mathbb{Z}_p^{\ell_z \times \lambda}$$
$$\mathsf{ct}_s = (\mathbf{h}, \mathbf{h}^s, \mathbf{h}^{s^2} \cdot g^{\overline{\mathbf{s}}}), \qquad \mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}^{-s} \cdot g^{\mathbf{\Delta}}).$$

Finally, sample PRF keys $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, $\mathsf{key}_2 \leftarrow \mathcal{K}_2$, $\mathsf{key}_3 \leftarrow \mathcal{K}_3$. Output $\mathsf{sk} = (\mathsf{pp}, \mathbf{h}, \mathbf{H}, \overline{\mathbf{s}}, \mathsf{key}_1, \mathsf{key}_2, \mathsf{key}_3)$, and $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1, \mathsf{key}_3)$.

Correctness, Efficiency, and Security. Correctness arguments are the same as Construction 2, except that every invokation of DDLog has a δ' correctness error. We have set $\delta' = \delta/O(|C|)$ such that the overall error probability from running DDLog $O(|C|)$ times is below δ as required. The scheme has the same asymptotic efficiency as Construction 2, i.e., with $|\mathsf{evk}| \leq \ell_z \cdot \mathrm{poly}(\lambda)$.

We state the following security lemma, whose proof is completely analogous to Lemma 7.

Lemma 6. *Under CP-DDH in prime-order groups, Construction 3 is secure.*

3.3 aHMACs From Damgård-Jurik

While the Damgård-Jurik encryption scheme (Construction 1) can be viewed as an instantiation of the NIDLS framework, we note that its particular structure allows a more convenient DDLog algorithm (Lemma 4):

$$\mathsf{DDLog}_{N,\zeta}(c^{\mathsf{sk} \cdot x + z}) \equiv \mathsf{sk} \cdot x \cdot y + \mathsf{DDLog}_{N,\zeta}(c^z) \mod N^\zeta,$$

where c is any ciphertext that decrypts to some value y, and sk is not a random exponent, but a fixed secret value $\mathsf{sk} = \varphi(N)$. We will use this DDLog variant on ciphertexts encrypting the

[8] The output length of F_3 is truncated to $\lceil \log(2B/\delta') \rceil$ as required by ϕ.

inverse of the secret key $\mathsf{sk}^{-1} \bmod N^\zeta$, which can effectively remove a factor of sk from any tag of the form $\sigma = \mathsf{sk} \cdot x + k$ over $\mathbb{Z}$: [9]

$$\begin{aligned} & \mathsf{ct}_s \leftarrow \mathsf{DJ.Enc}(\mathsf{pk}, 1/\mathsf{sk} \bmod N^\zeta), \\ \implies & \mathsf{DDLog}_{N,\zeta}(\mathsf{ct}_s^{\mathsf{sk}\cdot x+k}) \equiv \mathsf{sk} \cdot x \cdot \mathsf{sk}^{-1} + \mathsf{DDLog}_{N,\zeta}(\mathsf{ct}_s^k) \mod N^\zeta, \\ & \equiv x + \mathsf{DDLog}_{N,\zeta}(\mathsf{ct}_s^k) \mod N^\zeta. \end{aligned}$$

This leads to the following evaluation procedures for a Mult gate in $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$ respectively:

$\mathsf{EvalTag}$ given σ_x, σ_y computes:

$$a = \mathsf{ct}_x^{\sigma_x \cdot \sigma_y}, \quad \sigma_z^* := -\mathsf{DDLog}_{N,\zeta}(a) + \sigma_x \cdot y + \sigma_y \cdot x \bmod N^\zeta$$

$\mathsf{EvalKey}$ given k_x, k_y computes:

$$b = \mathsf{ct}_x^{k_x \cdot k_y}, \quad k_z^* := \mathsf{DDLog}_{N,\zeta}(b) \bmod N^\zeta$$

The DDLog algorithm ensures $\sigma_z^* = \mathsf{sk} \cdot x \cdot y + k_z^* \bmod N^\zeta$. The $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$ algorithms then apply a common random shift to σ_z^* and k_z^* respectively to make the equality holds also over $\mathbb{Z}$. We omit further details for this construction, which are analogous to Construction 2. Security of this construction relies on the KDM security of Damgård-Jurik encryption, which ensures ct_s does not leak anything about sk.

We note that in a recent concurrent work [MORS25], very similar techniques to the above are used to achieve a primitive called semi-private offline HSS (also assuming the KDM security of Damgård-Jurik encryption). The authors also give a leveled variant of their construction assuming only the standard DCR assumption. Adapting their leveled construction leads to a leveled aHMAC scheme assuming DCR. We omit re-creating this leveled construction here, and refer readers to [MORS25] for more details. Inspired by their leveled construction based on DCR, we present leveled constructions based on DDH in the NIDLS framework and in prime-order groups in Section 3.4.

3.4 Leveled aHMACs without Circular Security Assumptions

In this section, we show leveled variants that avoid the circular security assumption CP-DDH in Construction 2 and 3, at the cost of a larger evaluation key evk with size growing linearly with the depth bound D of evaluation circuits: $|\mathsf{evk}| = (\ell_z + D) \cdot \mathrm{poly}(\lambda)$. The leveled variants assume DDH in the NIDLS framework and prime-order groups respectively.

In the following, we focus on explaining the modifications to Construction 2 and prove it secure. The modifications to Construction 3 and the security proof are analogous.

Construction 4 (Leveled aHMACs from the NIDLS Framework). This construction relies on the same ingradients as Construction 2.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^D, \mathbf{\Delta})$: Compared to Construction 2, the only difference is that the ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ (Equation 4) are replaced with per-level ciphertexts $\mathsf{ct}^{(j)}$ for $j \in [D]$, and a final one ct_Δ computed as follows. First, sample *two* secret exponents per level

$$\forall j = 0, \ldots, D, \quad s_L^{(j)}, s_R^{(j)} \leftarrow [\ell].$$

[9] This usage of DDLog is inspired by the technique from [MORS24].

Then compute the ciphertexts $\{\mathsf{ct}^{(j)}\}$ and ct_Δ.

$$\begin{aligned}
\forall j \in [D], \qquad & \mathbf{h} = g^{\mathbf{r}}, \text{ for fresh } \mathbf{r} \leftarrow [\ell]^{2\ell_s}, \\
& \mathsf{ct}^{(j)} := (\mathbf{h}, \mathbf{h}^{s_L^{(j)}}, \mathbf{h}^{s_R^{(j)}}, \mathbf{h}^{s_L^{(j)} \cdot s_R^{(j)}} \cdot f^{\mathsf{Bits}(s_L^{(j+1)}, s_R^{(j+1)})}), \\
\text{for } j = D, \qquad & \mathbf{H} = g^{\mathbf{R}}, \text{ for fresh } \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, \\
& \mathsf{ct}_\Delta := (\mathbf{H}, \mathbf{H}^{-s_L^{(D)}} \cdot f^{\mathbf{\Delta}}).
\end{aligned}$$

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$: Compared to Construction 2, the only difference is that the secret vector $\overline{\mathbf{s}}$ used to be the bits of a global secret s, but now is the bits of the level-0 secrets:

$$\overline{\mathbf{s}} = \overline{\mathbf{s}}^{(0)} := \mathsf{Bits}(s_L^{(0)}, s_R^{(0)}).$$

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$: Compared to Construction 2, there are two overall differences:

- Each tag assigned to an intermediate wire w, of depth j and with value x, used to have the form $\overline{\mathbf{s}} \cdot x + \mathbf{k}^{(w)}$ for a global secret vector $\overline{\mathbf{s}}$, but now will have the form $\overline{\mathbf{s}}^{(j)} \cdot x + \mathbf{k}^{(w)}$ for a per-level secret vector $\overline{\mathbf{s}}^{(j)} := \mathsf{Bits}(s_L^{(j)}, s_R^{(j)})$.
- Before evaluating a gate, an additional step is required to ensure the tags on both input wires have the same-level secret vector.

We first present evaluation procedures for Add and Mult gates assuming both input tags $\sigma^{(w_1)}, \sigma^{(w_2)}$ have the same level-j secret vector.

- For Add gates, set $\sigma^{(w)} := \sigma^{(w_1)} + \sigma^{(w_2)}$ over $\mathbb{Z}^{2\ell_s}$.

> Note that if the input tags have the form $\sigma^{(w_1)} = \overline{\mathbf{s}}^{(j)} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \overline{\mathbf{s}}^{(j)} \cdot x_2 + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}$, then the computed tag also has the form $\sigma^{(w)} = \overline{\mathbf{s}}^{(j)} \cdot z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$, where $\mathbf{k}^{(w)} = \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$.

- For Mult gates, parse $\sigma^{(w_1)} = (\sigma_L^{(w_1)}, \sigma_R^{(w_1)})$, $\sigma^{(w_2)} = (\sigma_L^{(w_2)}, \sigma_R^{(w_2)})$, and $\mathsf{ct}^{(j)} = (\mathbf{h}, \mathbf{h}_{1,L}, \mathbf{h}_{1,R}, \mathbf{h}_2)$. Compute

$$\begin{aligned}
\mathbf{a}^{(w)} &:= \mathbf{h}^{\mathsf{BC}(\sigma_L^{(w_1)}) \cdot \mathsf{BC}(\sigma_R^{(w_2)})} \odot \mathbf{h}_{1,R}^{-\mathsf{BC}(\sigma_L^{(w_1)}) \cdot x_2} \odot \mathbf{h}_{1,L}^{-\mathsf{BC}(\sigma_R^{(w_2)}) \cdot x_1} \odot \mathbf{h}_2^{x_1 \cdot x_2}, \\
\sigma^{(w)} &:= \mathsf{DDLog}(\mathsf{pp}, \mathbf{a}^{(w)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \mod t.
\end{aligned}$$

> We show if the input tags have the form $\sigma^{(w_1)} = \overline{\mathbf{s}}^{(j)} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \overline{\mathbf{s}}^{(j)} \cdot x_2 + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}$, then the computed tag has the form $\sigma^{(w)} = \overline{\mathbf{s}}^{(j+1)} \cdot z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$, with the level-$(j+1)$ secret vector. We rely on the following core identity:
>
> $$\begin{aligned}
> &\mathsf{BC}(\sigma_L^{(w_1)})\mathsf{BC}(\sigma_R^{(w_2)}) - s_R^{(j)} \cdot \left(\mathsf{BC}(\sigma^{(w_1)})x_2\right) - s_L^{(j)} \cdot \left(\mathsf{BC}(\sigma^{(w_2)})x_1\right) + s_L^{(j)} s_R^{(j)} z \\
> =&\mathsf{BC}(\mathbf{k}_L^{(w_1)})\mathsf{BC}(\mathbf{k}_R^{(w_2)}) \quad \text{over } \mathbb{Z}.
> \end{aligned}$$

Plugging in $\mathbf{h}_{1,L} = \mathbf{h}^{s_L^{(j)}}$, $\mathbf{h}_{1,R} = \mathbf{h}^{s_R^{(j)}}$, and $\mathbf{h}_2 = \mathbf{h}^{s_L^{(j)} s_R^{(j)}} \cdot f^{\overline{\mathbf{s}}^{(j+1)}}$, we obtain

$$\begin{aligned}
&\mathbf{a}^{(w)} = f^{\overline{\mathbf{s}} \cdot z} \cdot \mathbf{h}^{\mathsf{BC}(\mathbf{k}_L^{(w_1)})\mathsf{BC}(\mathbf{k}_R^{(w_2)})} \\
\Rightarrow &\mathsf{DDLog}(\mathsf{pp}, \mathbf{a}^{(w)}) = \overline{\mathbf{s}} \cdot z + \mathsf{DDLog}(\mathsf{pp}, \mathbf{h}^{\mathsf{BC}(\mathbf{k}_L^{(w_1)})\mathsf{BC}(\mathbf{k}_R^{(w_2)})}) \bmod t. \\
\Rightarrow &\sigma^{(w)} = \overline{\mathbf{s}}^{(j+1)} \cdot z + \mathbf{k}^{(w)} \bmod t, \\
&\text{w/ } \mathbf{k}^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{h}^{\mathsf{BC}(\mathbf{k}_L^{(w_1)})\mathsf{BC}(\mathbf{k}_R^{(w_2)})}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \bmod t.
\end{aligned}$$

We have showed the desired invariant mod t. By the same argument as in Construction 2, it also holds over $\mathbb{Z}$ except with negligible probabiblity.

We can now transform any level-j tag to a level-$(j+1)$ tag of the same value by applying the described Mult procedure with another level-j tag of the constant value 1. We can obtain level-j tag of 1 for all levels, by starting from an arbitray input tag (of level-0) and squaring it j times.

$\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$: As in Construction 2, perform matching evaluations over MAC keys in the same order as $\mathsf{EvalTag}$. We present evaluation procedures for Add and Mult gates assuming the input MAC keys are $\mathbf{k}^{(w_1)}, \mathbf{k}^{(w_2)}$, and the output wire w has depth j.

- For Add gates, set $\mathbf{k}^{(w)} := \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}^{2\ell_s}$.

  As noted in $\mathsf{EvalTag}$, the matching evaluated tag equals $\sigma^{(w)} = \overline{\mathbf{s}}^{(j)} z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$.

- For Mult gates, parse $\mathbf{k}^{(w_1)} = (\mathbf{k}_L^{(w_1)}, \mathbf{k}_R^{(w_1)})$, and $\mathbf{k}^{(w_2)} = (\mathbf{k}_L^{(w_2)}, \mathbf{k}_R^{(w_2)})$. Read $\mathbf{h}^{(j)}$ from sk, and compute

$$\begin{aligned}
\mathbf{b}^{(w)} &:= \left(\mathbf{h}^{(j)}\right)^{\mathsf{BC}(\mathbf{k}_L^{(w_1)}) \cdot \mathsf{BC}(\mathbf{k}^{(w_2)_R})}, \\
\mathbf{k}^{(w)} &:= \mathsf{DDLog}(\mathsf{pp}, \mathbf{b}^{(w)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \mod t.
\end{aligned}$$

  As noted in $\mathsf{EvalTag}$, the matching evaluated tag equals $\sigma^{(w)} = \overline{\mathbf{s}}^{(j+1)} z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$.

Correctness, Efficiency, and Security. As before, we have broken up and embedded correctness analysis as notes in the above. Compared to Construction 2, the leveled construction has a larger evk consisting of per-level ciphertexts $\{\mathsf{ct}^{(j)}\}_{[D]}$ of $\mathrm{poly}(\lambda)$ bits each, and a final one ct_Δ of $\ell_z \cdot \mathrm{poly}(\lambda)$ bits. In total, the bit-length of evk is bounded by $(D + \ell_z) \cdot \mathrm{poly}(\lambda)$. Finally, we state and prove the following security lemma.

Lemma 7. *Under DDH in the NIDLS framework, Construction 4 is secure.*

Proof. The security of an aHMAC scheme (Definition 13) requires a pair of simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ to simulate an evaluation key evk and adaptively queried authentication tags σ.

- Sim_1 samples all components of the simulated $\mathsf{evk} = (\mathsf{pp}, \{\mathsf{ct}^{(j)}\}_{[D]}, \mathsf{ct}_\Delta, \mathsf{key}_1)$ at random. In more detail, it samples a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, public parameters of a NIDLS group

$\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, and a random group element $(g, \rho) \leftarrow \mathsf{Samp}(\mathsf{pp})$. It then samples random ciphertexts $\mathsf{ct}^{(j)} = (\widetilde{\mathbf{h}}^{(j)}, \widetilde{\mathbf{h}}_{1,L}^{(j)}, \widetilde{\mathbf{h}}_{1,R}^{(j)}, \widetilde{\mathbf{h}}_2^{(j)})$, and $\mathsf{ct}_\Delta = (\widetilde{\mathbf{H}}, \widetilde{\mathbf{H}}_1)$:

$$\begin{aligned} \forall j \in [D],\ \widetilde{\mathbf{h}}^{(j)} = g^{\mathbf{r}^{(j)}},\ \widetilde{\mathbf{h}}_{1,L}^{(j)} = g^{\mathbf{r}_{1,L}^{(j)}},\ \widetilde{\mathbf{h}}_{1,R}^{(j)} = g^{\mathbf{r}_{1,R}^{(j)}},\ \widetilde{\mathbf{h}}_2^{(j)} = g^{\mathbf{r}_2^{(j)}}, \quad & \mathbf{r}^{(j)}, \mathbf{r}_{1,L}^{(j)}, \mathbf{r}_{1,R}^{(j)}, \mathbf{r}_2^{(j)} \leftarrow [\ell]^{2\ell_s}, \\ \widetilde{\mathbf{H}} = g^{\mathbf{R}},\ \widetilde{\mathbf{H}}_1 = g^{\mathbf{R}_1} & \mathbf{R}, \mathbf{R}_1 \leftarrow [\ell]^{\ell_z \times \lambda}. \end{aligned}$$

– Sim_2 samples the authentication tag at random $\widetilde{\sigma} \leftarrow [2^\lambda]^{2\ell_s}$.

We show a series of hybrids that transitions from the real-world experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^0_{\mathrm{priv}}$ in Definition 13 to the simulation-world experiment $\mathsf{Hyb}_3 = \mathrm{Exp}^1_{\mathrm{priv}}$.

Hyb_0 : We summarize the real-world distribution of the evaluation key $\mathsf{evk} = (\mathsf{pp}, \{\mathsf{ct}^{(j)}\}, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}^{(j)} = (\mathbf{h}^{(j)}, \mathbf{h}_{1,L}^{(j)}, \mathbf{h}_{1,R}^{(j)}, \mathbf{h}_2^{(j)})$, and $\mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}_1)$, and of the authentication tag σ for some query (x, id).

$$\begin{aligned} & \mathsf{key}_1 \leftarrow \mathcal{K}_1, \quad \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda), && \left| \begin{array}{l} (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), \\ s_L^{(j)}, s_R^{(j)} \leftarrow [\ell],\ \forall j = 0, \ldots, D, \end{array} \right. \\ & \forall j \in [D] : \mathbf{h}^{(j)} = g^{\mathbf{r}^{(j)}},\ \mathbf{h}_{1,L}^{(j)} = g^{s_L^{(j)} \cdot \mathbf{r}^{(j)}},\ \mathbf{h}_{1,R}^{(j)} = g^{s_R^{(j)} \cdot \mathbf{r}^{(j)}}, && \left| \begin{array}{l} \mathbf{r}^{(j)} \leftarrow [\ell]^{2\ell_s}, \\ \overline{\mathbf{s}}^{(j+1)} := \mathsf{Bits}(s_L^{(j+1)}, s_R^{(j+1)}), \end{array} \right. \quad (7) \\ & \mathbf{h}_2^{(j)} = g^{s_L^{(j)} s_R^{(j)} \cdot \mathbf{r}^{(j)}} \cdot f^{\overline{\mathbf{s}}^{(j+1)}}, \\ & \mathbf{H} = g^{\mathbf{R}},\ \mathbf{H}_1 = g^{s_L^{(D)} \cdot \mathbf{R}} \cdot f^{\boldsymbol{\Delta}}, && \left| \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, \right. \quad (8) \\ & \sigma = \overline{\mathbf{s}}^{(0)} \cdot x + \mathsf{F}_2^{\ell_s}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathbb{Z} && \left| \mathsf{key}_2 \leftarrow \mathcal{K}_2, \right. \quad (9) \end{aligned}$$

Hyb_1 : Instead of computing each tag σ as in Equation 9, Hyb_1 simulates it as $\widetilde{\sigma} \leftarrow [2^\lambda]^{\ell_s}$. The PRF security of F_2 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

$\mathsf{Hyb}_{2,0,0}$: Instead of computing $\mathbf{h}^{(0)}, \mathbf{h}_{1,L}^{(0)}, \mathbf{h}_{1,R}^{(0)}, \mathbf{h}_2^{(0)}$ as in Equation 7, compute them from independent random exponents as follows:

$$\left| \begin{array}{l} \mathbf{h}^{(0)} = g^{\mathbf{r}^{(0)}},\ \mathbf{h}_{1,L}^{(0)} = g^{\mathbf{r}_{1,L}^{(0)}},\ \mathbf{h}_{1,R}^{(0)} = g^{\mathbf{r}_{1,R}^{(0)}}, \\ \mathbf{h}_2^{(0)} = g^{\mathbf{r}_2^{(0)}} \cdot f^{\overline{\mathbf{s}}^{(1)}}, \end{array} \right| \mathbf{r}^{(0)}, \mathbf{r}_{1,L}^{(0)}, \mathbf{r}_{1,R}^{(0)}, \mathbf{r}_2^{(0)} \leftarrow [\ell]^{2\ell_s}.$$

By DDH in the NIDLS framework (Definition 8), we have $\mathsf{Hyb}_{2,0,0} \approx_c \mathsf{Hyb}_1$.

$\mathsf{Hyb}_{2,0,1}$**:** Instead of computing $\mathbf{h}_2^{(0)}$ as the previous hybrid, directly compute it as

$$\mathbf{h}_2^{(0)} = g^{\mathbf{r}_2^{(0)}}, \ \left| \mathbf{r}_2^{(0)} \leftarrow [\ell]^{2\ell_s}. \right.$$

without depending on the secret vector $\overline{\mathbf{s}}^{(1)}$. The Samp algorithm (Definition 6) ensures

$$g^{\mathbf{r}_2^{(0)}} \cdot f^{\overline{\mathbf{s}}^{(1)}} \approx \mathsf{Uniform}(\langle g \rangle) \cdot f^{\overline{\mathbf{s}}^{(1)}} \equiv \mathsf{Uniform}(\langle g \rangle) \approx g^{\mathbf{r}_2^{(0)}}.$$

Hence we have $\mathsf{Hyb}_{2,0,1} \approx \mathsf{Hyb}_{2,0,0}$.

$\mathsf{Hyb}_{2,j}$: for $j = 1, \ldots D-1$, instead of computing $\mathbf{h}^{(j)}, \mathbf{h}_{1,L}^{(j)}, \mathbf{h}_{1,R}^{(j)}, \mathbf{h}_2^{(j)}$ as in Equation 7, compute them from independent random exponents as follows:

$$\left| \begin{array}{l} \mathbf{h}^{(j)} = g^{\mathbf{r}^{(j)}},\ \mathbf{h}_{1,L}^{(j)} = g^{\mathbf{r}_{1,L}^{(j)}},\ \mathbf{h}_{1,R}^{(j)} = g^{\mathbf{r}_{1,R}^{(j)}}, \\ \mathbf{h}_2^{(j)} = g^{\mathbf{r}_2^{(j)}} \end{array} \right| \mathbf{r}^{(j)}, \mathbf{r}_{1,L}^{(j)}, \mathbf{r}_{1,R}^{(j)}, \mathbf{r}_2^{(j)} \leftarrow [\ell]^{2\ell_s}.$$

By analogous arguments from $\mathsf{Hyb}_{2,0,0}$ and $\mathsf{Hyb}_{2,0,1}$, we have $\mathsf{Hyb}_{2,j} \approx_c \mathsf{Hyb}_{2,j-1}$.

Hyb_3 Instead of computing $\mathbf{H}, \mathbf{H}_1$ as in Equation 8, compute them from independent random exponents as follows:

$$\mathbf{H} = g^{\mathbf{R}}, \mathbf{H} = g^{\mathbf{R}_1} \Big| \mathbf{R}, \mathbf{R}_1 \leftarrow [\ell]^{\ell_z \times \lambda}.$$

By analogous arguments from $\mathsf{Hyb}_{2,0,0}$ and $\mathsf{Hyb}_{2,0,1}$, we have $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_{2,D-1}$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$, which proves the lemma. □

3.5 Standard HMAC from aHMAC

In this section, we explain the connection between our notion of aHMAC and standard HMAC schemes [AB09, GW13, CF13] (Definition 14). First, we show that with minor syntactical changes, we can adapt an aHMAC scheme into a HMAC scheme satisfying 1-hop verifiability (Definition 15) and a weak variant of unforgeability (Definition 16). Then, we show that our constructions of aHMAC can actually be modified to satsify multi-hop verifiability (Definition 18). (We keep the main construction for 1-hop evaluation as it's more convenient for the applications considered in this paper.) Finally, we show how to amplify weak unforgeability to standard unforgeability generically. The amplified scheme don't not have the algebraic form anymore.

Definition 14 (Homomorphic MAC (HMAC)). *A homomorphic MAC scheme has the a similar syntax to an aHMAC scheme (Definition 13), except with a slightly different* KeyGen *algorithm, and with a* Verify *instead of* $\mathsf{EvalKey}$ *algorithm.*

- $\mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$ *takes in an upperbound* ℓ_z *on the output length of supported evaluation circuits, and outputs a secret key* sk *and evaluation key* evk.
- $\mathsf{Auth}, \mathsf{EvalTag}$ *has the same syntax as Definition 13.*
- $\mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$: *takes as inputs the secret key* sk*, a Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$*, the* id*s associated with the inputs, output bits* $\mathbf{z} \in \{0,1\}^{\ell_z}$*, and evaluated tags* $\sigma_{\mathbf{z}} \in \mathbb{Z}^{\ell_z}$ *authenticating* $\mathbf{z}$*. It outputs either* $\top$*, indicating accept, or* $\bot$*, indicating reject.*

Remark 8. We define a leveled variant where KeyGen additionally takes a depth-bound D on evaluation circuits as 1^D, and require $\mathsf{EvalTag}$ to only take circuits of depth less than D.

Definition 15 ((1-hop) δ-Verifiability). *Let* $\delta = \delta(\lambda)$ *be an error bound. For every polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$ *where* $|C| \le p(\lambda)$*, inputs* $\mathbf{x} \in \{0,1\}^{\ell_x}$*, and ids* $\mathbf{id} \in \{0,1\}^{\ell_x \times \lambda}$*, the following holds*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, & (\mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z}) \\ \quad C(\mathbf{x}), \sigma_{\mathbf{z}}) = \top & \quad \sigma^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \mathbf{id}[i]) \\ & \quad \sigma_{\mathbf{x}} := (\ldots, \sigma^{(i)}, \ldots) \\ & \quad \sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}) \end{array}\right] \ge 1 - \delta(\lambda) - \mathrm{negl}(\lambda).$$

Definition 16 (Unforgeability). *For every efficient adversary* $\mathcal{A}$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, the following holds*

$$\Pr[\mathrm{Exp}^{\mathcal{A}}_{\mathrm{UF}}(\lambda) = \mathrm{Win}] \le \mathrm{negl}(\lambda),$$

where the experiment $\mathrm{Exp}_{\mathrm{UF}}$ *is as follows:*

1. *Launch* $\mathcal{A}(1^\lambda)$. *Receive from* $\mathcal{A}$ *an output length* 1^{ℓ_z}, *compute* $(\mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$, *and send* evk *to* $\mathcal{A}$.
2. *Receive from* $\mathcal{A}$ *any number of adaptively chosen queries in the following forms.*
   - *Authentication queries* $\{x^{(i)}, \mathsf{id}^{(i)}\}$ *with distinct* id*s.*
   - *Verification queries* $\{C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z}}^{(i)}\}$.

   > *In the weak version, we require* $\mathbf{id}^{(i)}$ *to contain only previously queried* id*s.*

   Answer them using $\mathsf{sk}, \boldsymbol{\Delta}$ *and running* $\mathsf{Auth}, \mathsf{Verify}$.
3. *Receive from* $\mathcal{A}$ *a forgery* $(C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$, *and output* Win *if* $\mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}}) = \top$, *and either of the following is true.*
   - *All* $id^{(i)} \in \mathbf{id}$ *are queried in Step 2. Let* $\mathbf{x}$ *be the queried inputs corresponding to* $\mathbf{id}$. *We have* $C(\mathbf{x}) \neq \mathbf{z}$.
   - *There exist* i *such that* $\mathbf{id}[i]$ *is not queried in Step 2.*

   > *In the weak version, we require* C *to not be determined by the already queried (partial) inputs. More specifically, we require the remaining positions can be efficiently assigned in two ways to cause differen evaluations of* C.

Remark 9. We embeded the definition of *weak* unforgeability in the above as boxed notes. We write $\mathrm{Exp}_{\mathrm{wUF}}$ to denote the weak unforgeability experiment. We will also consider the following slight variants to the experiments $\mathrm{Exp}_{\mathrm{UF}}, \mathrm{Exp}_{\mathrm{wUF}}$:

- We can enforce a bounded $Q(\lambda) \leq \mathrm{poly}(\lambda)$ number of verification queries from the adversary.
- We can allow a $\delta = 1/\mathrm{poly}(\lambda)$ chance of the adversary winning. We call it soundness error.

Theorem 7 (HMACs). *We have the following constructions:*

1. *Assuming CP-DDH in the NIDLS framework (e.g. Damgård-Jurik groups and class groups) or the KDM security of Damgård-Jurik encryption, there exists an HMAC scheme for arbitrary Boolean circuits, achieving 0-hop and multi-hop* negl*-verifiability (Definition 17, 18), and unforgeability (Definition 18).*
2. *Assuming CP-DDH in prime-order groups, for every polynomials* $p_1(\lambda), p_2(\lambda), p_3(\lambda)$, *there exists an HMAC scheme for arbitrary Boolean circuits, achieving 0-hop and multi-hop* $1/p_1$*-verifiability, and unforgeability assuming* $\leq p_2(\lambda)$ *verification queries and with* $1/p_3$ *soundness errors.*

In the above, an authentication tag (evaluated or not) costs $\mathrm{poly}(\lambda)$ *bits, and an* evk *costs* $\ell_z \cdot \mathrm{poly}(\lambda)$ *bits.*

Theorem 8 (Leveled HMACs). *We have the following constructions:*

1. *Assuming DDH in the NIDLS framework or the semantic security of Damgård-Jurik encryption (i.e. the DCR assumption), there exists a leveled HMAC scheme for bounded-depth Boolean circuits, achieving 0-hop and multi-hop* $\mathrm{negl}(\lambda)$*-verifiability and unforgeability.*
2. *Assuming DDH in prime-order groups, for every polynomials* $p_1(\lambda), p_2(\lambda), p_3(\lambda)$, *there exists a leveled aHMAC scheme for bounded-depth Boolean circuits, achieving 0-hop and multi-hop* $1/p_1$*-verifiability, and unforgeability assuming* $\leq p_2(\lambda)$ *verification queries and with* $1/p_3$ *soundness errors.*

In the above, an authentication tag (evaluated or not) costs $\mathrm{poly}(\lambda)$ *bits, and an* evk *supporting bounded-depth circuits by* D *costs* $(\ell_z + D) \cdot \mathrm{poly}(\lambda)$ *bits.*

In the remaining of this section, we focus on explaining the outlined steps to obtain non-leveled HMACs (Theorem 7) from aHMACs. The steps to obtain leveled HMACs (Theorem 8) from aHMACs are analogous.

Syntactical Changes from aHMAC to HMAC. The syntax of $\mathsf{Auth}, \mathsf{EvalTag}$ are the same for both aHMAC and HMAC. The differences are (1) in HMAC, the KeyGen algorithm doesn't take any user-supplied vector $\mathbf{\Delta}$ and (2) in HMAC a Verify algorithm replaces the $\mathsf{EvalKey}$ algorithm. To obatin a standard HMAC scheme from aHMAC, we modify KeyGen, and implement Verify as follows. ($\mathsf{Auth}, \mathsf{EvalTag}$ and are the same as $\mathsf{aHMAC.Auth}, \mathsf{aHMAC.EvalTag}$.)

Construction 5 (HMAC from aHMAC).

- $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$: Sample a global secret $\mathbf{\Delta} \leftarrow [2^\lambda]^{\ell_z}$, and run $(\mathsf{sk}', \mathsf{evk}') \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \mathbf{\Delta})$. Output $\mathsf{sk} := (\mathsf{sk}', \mathbf{\Delta})$ and $\mathsf{evk} = \mathsf{evk}'$.
- $b \leftarrow \mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$: First parse $\mathsf{sk} = (\mathsf{sk}', \mathbf{\Delta})$ and compute the evaluated MAC key $\mathbf{k}_C \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{sk}', C, \mathbf{id})$. Then check whether the evaluated tags satisfy the correct form: $\sigma_{\mathbf{z}} \stackrel{?}{=} \mathbf{\Delta} \odot \mathbf{z} + \mathbf{k}_C$ (over $\mathbb{Z}$). If yes, output $b := \top$. Otherwise, output $b := \bot$.

It's clear (hence we omit the proof) that if the underlying aHMAC scheme has δ-correctness, then the obtained HMAC scheme satisfies δ-verifiability.

Proposition 1. *Assuming the underlying aHMAC scheme has* δ*-correctness per Definition 13, then Construction 5 satisfies* δ*-verifiability per Definition 15.*

We show that the straightforwardly adapted HMAC scheme from any aHMAC scheme with negl correctness error satisfy *weak* unforgeability.

Proposition 2. *Assuming the underlying aHMAC scheme has* $\mathrm{negl}(\lambda)$*-correctness per Definition 13, then Construction 5 satisfy* weak *unforgeability per Definition 16.*

Proof. We show a series of hybrids that transitions from $\mathsf{Hyb}_0 := \mathrm{Exp}^{\mathcal{A}}_{\mathrm{wUF}}(\lambda)$ to Hyb_3, where the the queries of $\mathcal{A}$ are all answered without depending on the global secret $\mathbf{\Delta}$. We then argue that a forgery is impossible in Hyb_3 due to the randomness of $\mathbf{\Delta}$.

Hyb_0**:** This is the experiment $\mathrm{Exp}^{\mathcal{A}}_{\mathrm{wUF}}(\lambda)$.

Hyb_1**:** Instead of answering each verification query $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma^{(i)}_{\mathbf{z}})$ using $\mathsf{sk}, \mathbf{\Delta}$ and running Verify, proceed as follows.

- Let $\mathbf{x}, \sigma_{\mathbf{x}}$ be the already queried inputs and tags associated with $\mathbf{id}$. (By assumption, all $\mathsf{id} \in \mathbf{id}$ have been queried.)
- Compute $\sigma^*_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C^{(i)}, \mathbf{x}, \sigma_{\mathbf{x}})$, and answer according to the check

$$\sigma^*_{\mathbf{z}} + (\mathbf{z}^{(i)} - C^{(i)}(\mathbf{x})) \odot \mathbf{\Delta} = \sigma^{(i)}_{\mathbf{z}}. \tag{10}$$

By $\mathrm{negl}(\lambda)$-correctness, $\sigma_{\mathbf{z}}$ satisfy $\sigma^*_{\mathbf{z}} = \mathbf{\Delta} \odot C^{(i)}(\mathbf{x}) + \mathbf{k}_C$, while by construction, Verify only passes if $\sigma^{(i)}_{\mathbf{z}} = \mathbf{\Delta} \cdot \mathbf{z}^{(i)} + \mathbf{k}_C = \sigma^*_{\mathbf{z}} + (\mathbf{z}^{(i)} - C^{(i)}(\mathbf{x})) \odot \mathbf{\Delta}$. Therefore, we have $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_0$.

Hyb_2**:** Instead of checking the forgery $(C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$ by running Verify, proceed as follows.

- If every $id \in \mathbf{id}$ has been queried, with associated input $\mathbf{x}$, and $C(\mathbf{x}) = \mathbf{z}$, output "Loss";
- If exists an $id \in \mathbf{id}$ (at i-th position) not queried in Step 2, but C is already determined by the queried inputs, then directly output "Loss";

– Otherwise, for every un-queried input position j, one can assign assign a bit x_j and compute an associated tag $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, x_j, \mathbf{id}[j])$ such that the "completed" input $\mathbf{x}$ associated with $\mathbf{id}$ satsify $C(\mathbf{x}) \neq \mathbf{x}$. Let $\mathbf{x}$ be the completed inputs corresponding to $\mathbf{id}$, and $\sigma_{\mathbf{x}}$ be associated tags. Then compute $\sigma^*_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$, and check

$$\sigma^*_{\mathbf{z}} + (\mathbf{z} - C(\mathbf{x})) \odot \boldsymbol{\Delta} \stackrel{?}{=} \sigma_{\mathbf{z}}.$$

If not, output "Loss". Otherwise, output "Win".

Similarly to the previous hybrid, by negl(λ)-correctness we have $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_1$.

Hyb_3**:** Instead of answering authentication queries using $\boldsymbol{\Delta}, \mathsf{sk}$ and running Auth, the experiment Hyb_3 runs $\mathsf{Sim}_1, \mathsf{Sim}_2$ as guaranteed by Definition 13 to simulate the evaluation key evk, the answers to authentication queries, and the tags associated with un-queried "fake" inputs. The aHMAC security guarantees $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

Note that the global secret $\boldsymbol{\Delta}$ is now only used for answering verification queries as in Equation 10 and in the final check for forgery.

Hyb_4**:** Instead of answering verification queries using $\boldsymbol{\Delta}$ as in Equation 10, directly check if $\mathbf{z}^{(i)} - C^{(i)}(\mathbf{x}) \stackrel{?}{=} 0$. Due to the randomness of $\boldsymbol{\Delta}$, we have $\mathsf{Hyb}_4 \approx \mathsf{Hyb}_3$.

Note that the global secret $\boldsymbol{\Delta}$ is now only used for checking for forgery.

By a hybrid argument, we conclude that $\mathrm{Exp}^{\mathcal{A}}_{\mathrm{wUF}}(\lambda) \equiv \mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_4$. In Hyb_4, the adversary $\mathcal{A}$ wins only if it outputs a forgery $(C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$ such that $\sigma_{\mathbf{z}} = \sigma^*_{\mathbf{z}} + (C(\mathbf{x}) - \mathbf{z}) \odot \boldsymbol{\Delta}$. As noted, the adversary $\mathcal{A}$'s view is entirely independent of $\boldsymbol{\Delta}$. Hence in the case of $C(\mathbf{x}) - \mathbf{z} \neq \mathbf{0}$, the forgery has negligible chance of passing the checks due to the randomness of $\boldsymbol{\Delta}$. □

The above proof of weak unforgeability relies on negl(λ)-correctness of the underlying aHMAC scheme to transition from Hyb_0 to Hyb_1 and then from Hyb_1 to Hyb_2. In general, if the underlying aHMAC has δ-correctness, then the transition from Hyb_0 to Hyb_2 incurrs a $Q \cdot \delta$ soundness error, where Q is the number of verification queries by the adversary.

Proposition 3. *Assuming the underlying aHMAC scheme has δ-correctness per Definition 13, and a bound Q on the number of verification query in the weak unforgeability experiment, then Construction 5 satisfy weak unforgeability with $\delta \cdot Q$ soundness error.*

Therefore, if assuming a polynomially bounded number of verification queries Q in the weak unforgeability experiment, then we can choose a sufficiently small $\delta \leq 1/(\mathrm{poly}(\lambda) \cdot Q)$ for the underlying aHMAC scheme and prove weak unforgeability with $1/\mathrm{poly}(\lambda)$ soundness error, for any $\mathrm{poly}(\lambda)$. This is how we obtain the HMAC instantiation using prime-order groups in Theorem 7 and 8.

Achieving Composability. In the literature of HMACs, a desirable feature is to allow further evaluations by $\mathsf{EvalTag}$ on previously evaluated tags. We refer to this as multi-hop verifiability formally defined as follows.

Definition 17 ((0-hop) δ-Verifiability). *Let $\delta = \delta(\lambda)$ be an error bound. There exists a negligible function* negl(λ) *such that for every $\ell_z \in \mathbb{N}$, input $x \in \{0,1\}$ and $\mathsf{id} \in \{0,1\}^\lambda$, the following holds*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{sk}, \mathsf{ID}, \mathsf{id}, & (\mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z}) \\ \qquad x, \sigma'_x) = \top & \qquad \sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id}) \\ & \qquad \sigma'_x \leftarrow \mathsf{EvalTag}(\mathsf{evk}, \mathsf{ID}, x, \sigma_x) \end{array}\right] \geq 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

where $\mathsf{ID} : \{0,1\} \to \{0,1\}$ is the identity circuit.

Remark 10. 0-hop verifiability is trivially implied by 1-hop verifiability (Definition 15).

Definition 18 ((multi-hop) δ-Verifiability). *Let $\delta = \delta(\lambda)$ be an error bound. For every polynomial $p(\lambda)$, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every composed labeled Boolean circuits $C := g(f_1, \ldots, f_n)$, where where $g : \{0,1\}^n \to \{0,1\}^{\ell_z}$ and $|g| \leq p(\lambda)$, ids $\mathbf{id}_i \in \{0,1\}^{\ell_{x_i} \times \lambda}$ corresponding to f_i, and intermediate evaulation results $\mathbf{w} \in \{0,1\}^n$ and tags $\sigma_{\mathbf{w}} \in \mathbb{Z}^n$, the following implication holds with probability $\geq 1 - \delta(\lambda) - \mathrm{negl}(\lambda)$.*

Setup *Sample* $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$.
Premise *For* $i \in [n]$, $\mathsf{Verify}(\mathsf{sk}, f_i, \mathbf{id}_i, \mathbf{w}[i], \sigma_{\mathbf{w}}[i]) = \top$.
Conclusion

$$\mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, g(\mathbf{w}), \sigma_{\mathbf{z}}) = \top \;\middle|\; \begin{array}{l} \sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, g, \mathbf{w}, \sigma_{\mathbf{w}}), \\ \mathbf{id} := (\mathbf{id}_1, \ldots, \mathbf{id}_n). \end{array}$$

The aHMAC definition in Definition 13 doesn't support multi-hop evaluation because the authenticated tags and evaluated tags have different forms. More specifically, we will need the evaluated tags (of x) to have an algebraic form $\mathbf{\Delta}x + \mathbf{k}_x$, where $\mathbf{\Delta}$ is a user-supplied integer vector. This is defined to make the evaluated tags compatible for further evaluation as a wire label by a garbling scheme (with "free-XOR" style labels) or as a memory share by a HSS scheme.

However, we note that our constructions of aHMAC (Construction 2, 3, or the one sketched in Section 3.3) can be straightforwardly modified to support multi-hop evaluations. Roughly, our evaluation algorithm $\mathsf{EvalTag}$ proceeds in two steps:

1. Evaluate the circuit C over input tags using evk. Every intermediate evaluated tag (of value x) has a consistent form $\bar{\mathbf{s}}x + \mathbf{k}_x$, where $\bar{\mathbf{s}}$ is a global internal secret vector.
2. In the end, apply a "key-switching" step to transform the output tags (of value x) to have the format $\mathbf{\Delta}x + \mathbf{k}'_x$, where $\mathbf{\Delta}$ is the user-supplied vector encoded in evk.

To support multi-hop evaluations, we can move the second step out of $\mathsf{EvalTag}$, and only perform it during verification Verify. This ensures all evaluated tags remain their internal format, and can be further evaluated freely.

Claim 1. *With the described modification to the underlying* aHMAC *scheme, (Construction 2, 3, or Section 3.3) and assuming it has δ-correctness, then Construction 5 satsify both 0-hop δ-verifiability and multi-hop δ-verifiability.*

Note that weak unforgeability is not affected by this modification, because (1) the view of an adversary, consisting of un-evaluated tags and evk, does not change and (2) a forgery of a tag in the internal format can be publicly transformed into a forgery in the original format by applying the "key-switching" step.

Amplifying Weak to Standard Unforgeability. Finally, we show that we can generically combine two instances of HMAC schemes with weak unforgeability to satisfy standard unforgeability. We describe the construction below.

Construction 6 (HMAC from Weak to Standard Unforgeability). Ingredients:

– An HMAC scheme HMAC with weak unforgeability.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$: Setup two instances of HMAC:

$$(\mathsf{sk}_0, \mathsf{evk}_0) \leftarrow \mathsf{HMAC.KeyGen}(1^\lambda, 1^{\ell_z}), \quad (\mathsf{sk}_1, \mathsf{evk}_1) \leftarrow \mathsf{HMAC.KeyGen}(1^\lambda, 1^{\ell_z}),$$

and output $\mathsf{sk} := (\mathsf{sk}_0, \mathsf{sk}_1)$ and $\mathsf{evk} := (\mathsf{evk}_0, \mathsf{evk}_1)$.

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$: Parse $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1)$, and authenticate x using two instances of HMAC:

$$\sigma_0 \leftarrow \mathsf{HMAC.Auth}(\mathsf{sk}_0, x, \mathsf{id}), \quad \sigma_1 \leftarrow \mathsf{HMAC.Auth}(\mathsf{sk}_1, x, \mathsf{id}).$$

Output $\sigma_x := (\sigma_0, \sigma_1)$.

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$: Parse $\mathsf{evk} = (\mathsf{evk}_0, \mathsf{evk}_1)$, and evaluate two different circuits on $\sigma_{\mathbf{x}}$ using two instances of HMAC:

$$\sigma_{\mathbf{z},0} \leftarrow \mathsf{HMAC.EvalTag}(\mathsf{evk}_0, C, \mathbf{x}, \sigma_{\mathbf{x}}),$$
$$\sigma_{y,1} \leftarrow \mathsf{HMAC.EvalTag}(\mathsf{evk}_1, \mathsf{XOR}, \mathbf{x}, \sigma_{\mathbf{x}}),$$

where $y = \mathsf{XOR}(\mathbf{x})$ computes the XOR of all bits in $\mathbf{x}$. Output $\sigma_{\mathbf{z}} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,1})$

$\mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$: Parse $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1)$, and $\sigma_{\mathbf{z}} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,1})$. Then verify the evaluated tags $\sigma_{\mathbf{z},0}$ and $\sigma_{y,1}$ using two instances of HMAC:

$$b_0 \leftarrow \mathsf{HMAC.Verify}(\mathsf{sk}_0, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z},0}),$$
$$b_1 \leftarrow \mathsf{aHMAC.Verify}(\mathsf{sk}_1, \mathsf{XOR}, \mathbf{id}, y, \sigma_{y,1}).$$

Output $\top$ only if $b_0 = b_1 = \top$. Otherwise, output $\bot$.

The verifiability correctness of Construction 6 follows directly from that of the underlying HMAC scheme. We now show that Construction 6 satsify unforgeability per Definition 16.

Proposition 4. *Assuming the underlying HMAC scheme satsify weak unforgeability per Definition 16, then Construction 6 satsify (standard) unforgeability per Definition 16.*

Proof. We now show a series of hybrids that transitions from $\mathsf{Hyb}_0 := \mathrm{Exp}^{\mathcal{A}}_{\mathrm{UF}}(\lambda)$ to Hyb_2, where the probability of the experiment outputs "Win" is negligible.

Hyb_0**:** This is the experiment $\mathrm{Exp}^{\mathcal{A}}_{\mathrm{UF}}(\lambda)$.

Hyb_1**:** Instead of answering each verification query $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma^{(i)}_{\mathbf{z}})$ using sk and Verify, proceed as follows.

- If there is an $\mathsf{id} \in \mathbf{id}^{(i)}$ that's un-queried, directly answer $\bot$.
- Otherwise, run Verify to answer as in Hyb_0.

Note that Hyb_1 is the same as Hyb_0, unless there is a verification query with unqueried $\mathbf{id}$. We claim that this bad event happens with negligible probability, due to the weak unforgeability of (the second instance of) HMAC.

Claim. If the underlying HMAC scheme satisfy weak unforgeability, then in Hyb_0 all verification queries with an un-queried id evaluate to $\bot$, except with negligible probability.

We have $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Instead of checking the forgery $(C, \mathbf{id}, \mathbf{z}, \sigma_z)$ by running Verify, proceed as follows.

- If there is an $\mathsf{id} \in \mathbf{id}^{(i)}$ that's un-queried, directly answer "Loss".
- Otherwise, run Verify to answer as in Hyb_1.

By analogous arguments as the previous hybrid, we have $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_1$ due to the weak unforgeability of (the second instance of) HMAC.

By a hybrid argument, we conclude that $\mathrm{Exp}^{\mathcal{A}}_{\mathrm{UF}}(\lambda) \equiv \mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_2$. We claim that the probability of Hyb_2 outputing "Win" is negligible due to the weak unforgeability of (the first instance of) HMAC, which concludes the proof.

Claim. If the underlying HMAC scheme satisfy weak unforgeability, then Hyb_2 outputs "Loss" except with negligible probability.

We now prove the first claim.

Proof (of the first claim). For contradiction, assume there exists an efficient adversary $\mathcal{A}$ that with non-negligible probability $p(\lambda)$ produces at least one verification query in the Hyb_0 experiment, $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z}}^{(i)})$, where $\mathbf{id}^{(i)}$ contains an un-queried id, but Verify outputs $\top$. We call such queries "bad" ones. Then we define the following reduction $\mathcal{B}$ to break the weak unforgeability of the underlying HMAC scheme.

- Let $Q \leq \mathrm{poly}(\lambda)$ be a bound on the number of verification queries from $\mathcal{A}$, and $j \leftarrow [Q]$ be $\mathcal{B}$'s guess of the first "bad" query from $\mathcal{A}$.
- Recall that in Construction 5, authentication tags and verification queries are handled using two independent instances of HMAC schemes. $\mathcal{B}$ sets up the first instance on its own by running
$$(\mathsf{sk}_0, \mathsf{evk}_0) \leftarrow \mathsf{HMAC}(1^\lambda, 1^{\ell_z}).$$
It queries the challenger to handle the second instance of HMAC as we explain below.
- For every authentication query $(x^{(i)}, \mathsf{id}^{(i)})$, $\mathcal{B}$ computes $\sigma_0 \leftarrow \mathsf{HMAC.Auth}(\mathsf{sk}_0, x^{(i)}, \mathsf{id}^{(i)})$ on its own, and query the challenger $(x^{(i)}, \mathsf{id}^{(i)})$ to obtain σ_1. $\mathcal{B}$ answers $\sigma_x := (\sigma_0, \sigma_1)$ to $\mathcal{A}$.
- For every authentication query $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z}}^{(i)} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,0}))$, $\mathcal{B}$ checks if $\mathbf{id}^{(i)}$ contains an un-queried id.
  - If yes, and if $i < j$, then $\mathcal{B}$ aborts.
  - If yes, and if $i = j$, then $\mathcal{B}$ outputs $(\mathsf{XOR}, \mathbf{id}^{(i)}, y, \sigma_{y,0})$ as the forgery to the challenger.
  - If no, then $\mathcal{B}$ computes $b_0 \leftarrow \mathsf{HMAC.Verify}(\mathsf{sk}_0, C^{(i)}, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z},0})$, and query the challenger with $(\mathsf{XOR}, \mathbf{id}^{(i)}, y, \sigma_{y,0})$ to obtain b_1. $\mathcal{B}$ answers $\top$ to $\mathcal{A}$ if $b_0 = b_1 = \top$, and $\bot$ otherwise.

Note that if $\mathcal{B}$ guess correctly the index of the first "bad" query from $\mathcal{A}$, then it successfully wins the weak unforgeability experiment. This happens with non-negligible probability $\geq p(\lambda)/Q$. Hence we are done. □

We next prove the second claim.

Proof (of the second claim). For contradiction, assume there exists an efficient adversary $\mathcal{A}$ that wins the Hyb_2 experiment with non-negligible probability $p(\lambda)$. Then we define the following reduction $\mathcal{B}$ to break the weak unforgeability of the underlying aHMAC scheme.

- Recall that in Construction 5, authentication tags and verification queries are handled using two independent instances of HMAC schemes. $\mathcal{B}$ sets up the second instance on its own by running
$$(\mathsf{sk}_1, \mathsf{evk}_1) \leftarrow \mathsf{HMAC}(1^\lambda, 1^{\ell_z}).$$
It queries the challenger to handle the first instance of HMAC as we explain below.
- For every authentication query $(x^{(i)}, \mathsf{id}^{(i)})$, $\mathcal{B}$ computes $\sigma_1 \leftarrow \mathsf{HMAC.Auth}(\mathsf{sk}_1, x^{(i)}, \mathsf{id}^{(i)})$ on its own, and query the challenger $(x^{(i)}, \mathsf{id}^{(i)})$ to obtain σ_0. $\mathcal{B}$ answers $\sigma_x := (\sigma_0, \sigma_1)$ to $\mathcal{A}$.
- For every authentication query $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z}}^{(i)} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,0}))$, $\mathcal{B}$ checks if $\mathbf{id}^{(i)}$ contains an un-queried id.
  - If yes, answer $\bot$ to $\mathcal{A}$ as in Hyb_2.
  - If no, then $\mathcal{B}$ computes $b_1 \leftarrow \mathsf{HMAC.Verify}(\mathbf{\Delta}, \mathsf{sk}_0, \mathsf{XOR}, \mathbf{id}, y, \sigma_{y,1})$, and query the challenger with $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z},0})$ to obtain b_0. $\mathcal{B}$ answers $\top$ to $\mathcal{A}$ if $b_0 = b_1 = \top$, and $\bot$ otherwise.

– Given a forgery $(C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,1}))$ from $\mathcal{A}$, $\mathcal{B}$ checks if $\mathbf{id}$ contains un-queried ids.
  - If yes, then abort.
  - If no, then output $(C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z},0})$ to the challenger as a forgery.

Note that if $\mathcal{A}$ wins in the emulated experiment, then $\mathcal{B}$ also wins in the weak unforgeability experiment. By assumption, this happens with non-negligible probability $p(\lambda)$. Hence we are done. □

4 Succinct Partial Garbling

In this section, we show how to construct succinct partial garbling schemes for circuits from aHMACs. In Section 4.1, we show the simpler case using an aHMAC with negl correctness errors. In Section 4.2, we show the more general case using an aHMAC with 1/poly correctness errors, and a robust secret sharing scheme, e.g., Shamir's scheme.

Theorem 9 (Succinct Partial Garbling for Circuits). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be the class of two-input Boolean circuits, i.e.*

$$\mathcal{C}_\lambda := \{\textit{all Boolean circuits of form } C(\mathbf{x}, \mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}), \mathbf{y})\}.$$

There exists a succinct partial garbling scheme for $\mathcal{C}$ where the garbling size is $|\widehat{C}| = |C_{\text{priv}}| \cdot \text{poly}(\lambda)$ bits under any of the assumptions from Theorem 5:

1. *CP-DDH in either the NIDLS framework or prime-order groups;*
2. *the KDM-DCR assumption.*

When replacing the aHMAC with a leveled aHMAC in the above constructions, we obtain a partial garbling whose size scales linearly with both the private computation complexity $|C_{\text{priv}}|$ and the public computation depth D_{pub}. This scheme satisfies our succinctness definition (Definition 2) when restricted to the class of bounded-depth circuits. As the constructions remain unchanged otherwise, we omit writing them out again.

Theorem 10 (Succinct Partial Garbling for Bounded-Depth Circuits). *Let $\mathcal{C}$ be the class of two-input Boolean circuits as in Theorem 9. There exists a partial garbling scheme for $\mathcal{C}$ with garbling size $|\widehat{C}| = (|C_{\text{priv}}| + D_{\text{pub}}) \cdot \text{poly}(\lambda)$ bits, where D_{pub} denotes the depth of C_{pub}, under any of the assumptions from Theorem 6:*

1. *DDH in the NIDLS framework or prime-order groups.*
2. *The DCR assumption.*

As a directly implication, for any polynomial $d(\lambda)$, let $\mathcal{C}^d := \{\mathcal{C}^d_\lambda\}$ be the class of bounded-depth two-input Boolean circuits, i.e.

$$\mathcal{C}^d_\lambda := \{\textit{all Boolean circuits of form } C(\mathbf{x}, \mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}), \mathbf{y}),\ \textit{and with depth } \leq d(\lambda)\}.$$

There exists a succinct *partial garbling scheme for $\mathcal{C}^d$ with garbling size $|\widehat{C}| = (|C_{\text{priv}}| + d(\lambda)) \cdot \text{poly}(\lambda)$ bits, under the above assumptions.*

4.1 Construction from negl-Correct aHMACs

Construction 7 (Succinct Partial Garbling). Ingredients:

- An aHMAC scheme aHMAC with negl-correctness error.
- Any Boolean garbling scheme BG with λ-bit labels.
- A secret-key encryption scheme E with λ-bit keys encrypting λ-bit messages.

We construct a succinct partial garbling scheme for the class of Boolean circuits of unbounded size: $\mathcal{C} = \{\mathcal{C}_\lambda\}$, where every $\mathcal{C}_\lambda$ contains all Boolean circuits of the form $C(\mathbf{x}, \mathbf{y}) = C_{\mathrm{priv}}(C_{\mathrm{pub}}(\mathbf{x}), \mathbf{y})$. We refer to $C_{\mathrm{pub}} : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_w}$ as the public sub-circuit, and $C_{\mathrm{priv}} : \{0,1\}^{\ell_w} \times \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_z}$, the private sub-circuit.

$(\widehat{C}, \{K_x^{(i)}\}, \{K_y^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, C, \{K_z^{(i)}\})$:

1. Generate the garbling $\widehat{C_{\mathrm{priv}}}$ of the private sub-circuit:

$$(\widehat{C_{\mathrm{priv}}}, \{K_w^{(i)}\}, \{K_y^{(i)}\}) \leftarrow \mathsf{BG.Garb}(C_{\mathrm{priv}}, \{K_z^{(i)}\}).$$

2. Sample ℓ_w pairs of secret keys $\{\Delta_j, \overline{\Delta}_j\}$ to encrypt output labels of C_{pub}:

$$\begin{aligned} \text{for } j \in [\ell_w], \quad \mathsf{ct}_j &\leftarrow \mathsf{E.Enc}(\Delta_j, K_w^{(j)}(1)) \\ \overline{\mathsf{ct}}_j &\leftarrow \mathsf{E.Enc}(\overline{\Delta}_j, K_w^{(j)}(0)). \end{aligned}$$

3. Generate aHMAC tags for public inputs as their labels (using deterministically derived distinct $\mathbf{id}, \overline{\mathbf{id}}$):

$$\begin{aligned} (\mathsf{sk}, \mathsf{evk}) &\leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \mathbf{\Delta}), & \mathbf{\Delta} &:= (\ldots, \Delta_j, \ldots), \\ (\overline{\mathsf{sk}}, \overline{\mathsf{evk}}) &\leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \overline{\mathbf{\Delta}}), & \overline{\mathbf{\Delta}} &:= (\ldots, \overline{\Delta}_j, \ldots), \\ \sigma_b^{(i)} &\leftarrow \mathsf{aHMAC.Auth}(\mathsf{sk}, b, \mathbf{id}[i]), & &// \text{ for } i \in [\ell_x], b \in \{0,1\}, \\ \overline{\sigma}_b^{(i)} &\leftarrow \mathsf{aHMAC.Auth}(\overline{\mathsf{sk}}, b, \overline{\mathbf{id}}[i]). \end{aligned}$$

Define $K_x^{(i)}$ such that $K_x^{(i)}(b) = (\sigma_b^{(i)}, \overline{\sigma}_b^{(i)})$.

4. Evaluate aHMAC keys $\mathbf{k}_{\mathrm{pub}}, \overline{\mathbf{k}}_{\mathrm{pub}}$ which will be used as "decryption helpers":

$$\mathbf{k}_{\mathrm{pub}} \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{sk}, C_{\mathrm{pub}}, \mathbf{id}), \quad \overline{\mathbf{k}}_{\mathrm{pub}} \leftarrow \mathsf{aHMAC.EvalKey}(\overline{\mathsf{sk}}, \overline{C_{\mathrm{pub}}}, \overline{\mathbf{id}}),$$

where $\overline{C_{\mathrm{pub}}}$ computes the complement of C_{pub}.

Output $\widehat{C} := (\widehat{C_{\mathrm{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \mathsf{evk}, \overline{\mathsf{evk}}, \mathbf{k}_{\mathrm{pub}}, \overline{\mathbf{k}}_{\mathrm{pub}})$, $\{K_x^{(i)}\}$ and $\{K_y^{(i)}\}$.

$\mathbf{z} \leftarrow \mathsf{Eval}(C, \widehat{C}, \{x^{(i)}, L_x^{(i)}\}, \{L_y^{(i)}\})$:

Parse $\widehat{C_{\mathrm{priv}}}$, $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$, and $\mathsf{evk}, \overline{\mathsf{evk}}, \mathbf{k}_{\mathrm{pub}}, \overline{\mathbf{k}}_{\mathrm{pub}}$ from $\widehat{C}$. Let $\mathbf{x} := (\ldots, x_i, \ldots)$. Parse $\{\sigma^{(i)}, \overline{\sigma}^{(i)}\}$ from $\{L_x^{(i)}\}$, and let

$$\sigma_{\mathbf{x}} := (\ldots, \sigma^{(i)}, \ldots)_{i \in [\ell_x]}, \quad \overline{\sigma}_{\mathbf{x}} := (\ldots, \overline{\sigma}^{(i)}, \ldots)_{i \in [\ell_x]}.$$

1. Evaluate aHMAC tags according to C_{pub}:

$$\sigma_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{evk}, C_{\mathrm{pub}}, \mathbf{x}, \sigma_{\mathbf{x}}), \ \overline{\sigma}_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\overline{\mathsf{evk}}, \overline{C_{\mathrm{pub}}}, \mathbf{x}, \overline{\sigma}_{\mathbf{x}}).$$

Note that the aHMAC correctness should ensure $\sigma_{\mathbf{w}} = \mathbf{\Delta} \odot \mathbf{w} + \mathbf{k}_{\mathrm{pub}}$, *and* $\overline{\sigma}_{\mathbf{w}} = \overline{\mathbf{\Delta}} \odot \overline{\mathbf{w}} + \overline{\mathbf{k}}_{\mathrm{pub}}$ *over* $\mathbb{Z}$, *where* $\mathbf{w} = C_{\mathrm{pub}}(\mathbf{x})$, *and* $\overline{\mathbf{w}} = \mathbf{1} - \mathbf{w} = \overline{C_{\mathrm{pub}}}(\mathbf{x})$.

2. Recover one of the decryption keys Δ_j, $\overline{\Delta}_j$ for each bit of $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$.

$$\begin{aligned} &\text{If } \mathbf{w}[j] = 1 && \Delta_j \leftarrow \sigma_{\mathbf{w}}[j] - \mathbf{k}_{\text{pub}}[j] \quad \text{(over } \mathbb{Z}\text{)}, \\ &\text{o/w} && \overline{\Delta}_j \leftarrow \overline{\sigma}_{\mathbf{w}}[j] - \overline{\mathbf{k}}_{\text{pub}}[j] \quad \text{(over } \mathbb{Z}\text{)}. \end{aligned}$$

3. Decrypt input labels $\{L_w^{(i)}\}$ to the private sub-circuit C_{priv}:

$$\begin{aligned} &\text{If } \mathbf{w}[j] = 1 && L_w^{(j)} \leftarrow \mathsf{E.Dec}(\Delta_j, \mathsf{ct}_j), \\ &\text{o/w} && L_w^{(j)} \leftarrow \mathsf{E.Dec}(\overline{\Delta}_j, \overline{\mathsf{ct}}_j). \end{aligned}$$

Then evaluate the garbling $\{L_z^{(i)}\} \leftarrow \mathsf{BG.Eval}(C_{\text{priv}}, \widehat{C_{\text{priv}}}, \{L_w^{(i)}\}, \{L_y^{(i)}\})$.

Correctness: As noted in the construction, correctness follows straightforwardly from that of the aHMAC scheme and Boolean garbling.

Efficiency: We summarize bit-lengths of the components, assuming the Boolean garbling scheme BG has label size $O(\lambda)$ bits and garbling size $|C_{\text{priv}}| \cdot \text{poly}(\lambda)$ bits, the encryption scheme E has $O(1)$-rate ciphertexts, and the aHMAC scheme aHMAC has evaluation keys of $\ell_z \cdot \text{poly}(\lambda)$ bits. They all exist under any of the assumptions from which we construct aHMACs in Theorem 5.

- $\{L_x^{(i)}\}$ each consists of two aHMAC tags, which has $\text{poly}(\lambda)$ bits.
- $\{L_y^{(i)}\}$ are standard Boolean garbling labels, and each has $O(\lambda)$ bits.
- $\widehat{C}$ consists of the following, and has $|C_{\text{priv}}| \cdot \text{poly}(\lambda)$ bits overall.
  - A Boolean garbling $\widehat{C_{\text{priv}}}$, which has $|C_{\text{priv}}| \cdot O(\lambda)$ bits;
  - $2\ell_z$ ciphertexts $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$, each having $O(\lambda)$ bits;
  - 2 aHMAC evaluation keys $\mathsf{evk}, \overline{\mathsf{evk}}$, each having $\ell_z \cdot \text{poly}(\lambda)$;
  - 2 aHMAC evaluated keys $\mathbf{k}_{\text{pub}}, \mathbf{k}_{\overline{\text{pub}}}$, each having $\ell_z \cdot \text{poly}(\lambda)$.

Security: We will next show a more general construction from aHMACs with 1/poly correctness error. We omit proving security of the current simpler case, and refer readers to the more general proof (of Lemma 9).

Lemma 8. *Construction 7 is a secure partial garbling scheme.*

4.2 Construction from 1/poly-Correct aHMACs

Definition 19 (t-out-of-n Robust Secret Sharing). *A t-out-of-n robust secret sharing scheme with message space $\mathcal{M}$ consists of two efficient algorithms:*

- $\mathsf{Share}(s \in \mathcal{M})$ *takes a secret s, and outputs n shares $\{s_i\}_{[n]}$ where $s_i \in \mathcal{M}$.*
- $\mathsf{Recon}(\{s_i\}_{[n]})$ *takes n shares, and recovers a secret $s \in \mathcal{M}$.*

Robust Reconstruction. *For all messages $s \in \mathcal{M}$, all subsets $T \subset [n]$ with $|T| \leq t$, and any adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\mathsf{Recon}(\{s_i^*\}_{[n]}) = s \;\middle|\; \begin{array}{c} \{s_i\} \leftarrow \mathsf{Share}(s), \\ \{s_i^*\}_T \leftarrow \mathcal{A}(\{s_i\}_{[n]}), \\ s_i^* = s \text{ for } i \notin T. \end{array}\right] = 1$$

Privacy. *For any two messages $s, s' \in \mathcal{M}$, all subsets $T \subset [n]$ with $|T| \leq t$, the following holds:*

$$\mathsf{Share}(s)_T \equiv \mathsf{Share}(s')_T$$

Remark 11. For $t < n/3$, the standard Shamir's secret sharing is also a robust secret sharing. This suffices for our application. For $n/3 \le t < n/2$, there also exists robust secret sharing schemes with a negligible correctness error and larger share size, e.g. [RB89, CFOR12].

Construction 8 (Correctness and Privacy Amplification). Ingredients:

- An aHMAC scheme aHMAC with $1/(2\lambda)$-correctness error.
- A $(\lambda-1)$-out-of-3λ RSS scheme RSS with message space $\mathcal{M} = \{0,1\}^\lambda$.
- Any Boolean garbling scheme BG with λ-bit labels.
- A secret-key encryption scheme E with λ-bit keys encrypting λ-bit messages.

Compared to Construction 7, the new construction for Garb differs in Step 3, 4, and for Eval differs in Step 1, 2. In the following, we focus on the differences.

$(\widehat{C}, \{K_x^{(i)}\}, \{K_y^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, C)$:

3'. Generate 3λ shares $\{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_{t\in[3\lambda]}$ from each pairs of secret keys:

$$\text{for } j \in [\ell_w], \qquad \{\Delta_{j,t}\}_{t\in[3\lambda]} \leftarrow \mathsf{Share}(\Delta_j), \quad \{\overline{\Delta}_{j,t}\}_{t\in[3\lambda]} \leftarrow \mathsf{Share}(\overline{\Delta}_j).$$

Write $\mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_{j\in[\ell_w]}$ to denote the t-th share of the key pairs.

4'. Apply Step 3, 4 from Construction 7 independently on each share Keys_t to generate

$$\mathsf{evk}_t, \overline{\mathsf{evk}}_t, \mathbf{k}_{\mathrm{pub},t}, \overline{\mathbf{k}}_{\mathrm{pub},t}, \quad \text{and for } i \in [\ell_x],\ K_{x,t}^{(i)}.$$

Define $K_x^{(i)}$ such that $K_x^{(i)}(b) = (\ldots, K_{x,t}^{(i)}(b), \ldots)_{t\in[3\lambda]}$, and as shorthands write $\mathsf{Tables}_t = (\mathsf{evk}_t, \overline{\mathsf{evk}}_t, \mathbf{k}_{\mathrm{pub},t}, \overline{\mathbf{k}}_{\mathrm{pub},t})$.

Output $\widehat{C} := (\widehat{C_{\mathrm{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \{\mathsf{Tables}_t\})$, $\{K_x^{(i)}\}$ and $\{K_y^{(i)}\})$.

$\mathbf{z} \leftarrow \mathsf{Eval}(C, \widehat{C}, \{x^{(i)}, L_x^{(i)}\}, \{L_y^{(i)}\})$:

Parse $\widehat{C_{\mathrm{priv}}}$, $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$, and $\{\mathsf{Tables}_t\}$ from $\widehat{C}$. Let $\mathbf{x} := (\ldots, x_i, \ldots)$. Parse $\{\sigma_t^{(i)}, \overline{\sigma}_t^{(i)}\}$ from $\{L_x^{(i)}\}$, and let

$$\sigma_{\mathbf{x},t} := (\ldots, \sigma_t^{(i)}, \ldots)_{i\in[\ell_x]}, \quad \overline{\sigma}_{\mathbf{x},t} := (\ldots, \overline{\sigma}_t^{(i)}, \ldots)_{i\in[\ell_x]}.$$

1'. For each $t \in [3\lambda]$, apply Step 1, 2 from Construction 7 independently on each Tables_t to recover (half of) the t-th share $\mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}$ for each bit of $\mathbf{w} = C_{\mathrm{pub}}(\mathbf{x})$.

$$\text{If } \mathbf{w}[j] = 1, \quad \{\Delta_{j,t}\}_{t\in[3\lambda]}, \quad \text{o/w}, \quad \{\overline{\Delta}_{j,t}\}_{t\in[3\lambda]}.$$

Note that due to the $1/(2\lambda)$-correctness error from aHMAC, some of the recovered shares (but less than λ of them) may be incorrect. But this suffices for (robust) reconstruction of the secret sharing scheme.

2'. Apply robust secret share reconstruction to recover the decryption keys:

$$\text{If } \mathbf{w}[j] = 1, \quad \Delta_j \leftarrow \mathsf{Recon}(\{\Delta_{j,t}\}), \quad \text{o/w}, \quad \overline{\Delta}_j \leftarrow \mathsf{Recon}(\{\overline{\Delta}_{j,t}\}).$$

Then continue as in Construction 7 using the recovered decryption keys.

Correctness: As noted, correctness follows from the robust reconstruction property of the secret sharing scheme, which is able to tolerate up to $\lambda - 1$ incorrect shares.

Efficiency: Compared to Construction 7, the label sizes remain $\mathrm{poly}(\lambda)$ bits each, while the garbled circuit size is increased by at most 3λ times. We still have $|\widehat{C}| = |C_{\mathrm{priv}}| \cdot \mathrm{poly}(\lambda)$.

Security: We state and prove the following security lemma.

Lemma 9. *Construction 8 is a secure partial garbling scheme.*

Proof (of Lemma 9). The security of a partial garbling scheme (Definition 1) requires a simulator Sim, given a two-input circiut C, a public input $\mathbf{x}$, and output labels $\{L_z^{(i)}\}$, to simulate a garbled circuit $\widehat{C}$ and input labels $\{L_x^{(i)}\}$, $\{L_y^{(i)}\}$. It simulates them as follows:

- Run the (standard) Boolean garbling simulator $\mathsf{BG.Sim}$ to compute

$$\widetilde{C_{\mathrm{priv}}}, \{\widetilde{L}_w^{(i)}\}, \{\widetilde{L}_y^{(i)}\} \leftarrow \mathsf{BG.Sim}(1^\lambda, C_{\mathrm{priv}}, \{L_z^{(i)}\}).$$

- Sample random encryptions keys $\boldsymbol{\Delta} = (\ldots, \Delta_j, \ldots)_j$, $\overline{\boldsymbol{\Delta}} = (\ldots, \overline{\Delta}_j, \ldots)_{j \in [\ell_w]}$, and simulate ciphertexts $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}_{j \in [\ell_w]}$ of labels $L_w^{(j)}$ according to $\mathbf{w} = C_{\mathrm{pub}}(\mathbf{x})$.

$$\mathsf{ct}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\Delta_j, L_w^{(j)}) \\ \mathsf{E.Enc}(\Delta_j, 0) \end{cases} \qquad \overline{\mathsf{ct}}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\overline{\Delta}_j, 0) & \text{if } \mathbf{w}[j] = 1 \\ \mathsf{E.Enc}(\overline{\Delta}_j, L_w^{(j)}) & \text{o/w .} \end{cases} \tag{11}$$

- Secret share the encryption keys according to $\mathbf{w}$:

$$\{\Delta_{j,t}\}_t \leftarrow \begin{cases} \mathsf{Share}(\Delta_j), \\ \mathsf{Share}(0), \end{cases} \qquad \{\overline{\Delta}_{j,t}\}_t \leftarrow \begin{cases} \mathsf{Share}(0) & \text{if } \mathbf{w}[j] = 1, \\ \mathsf{Share}(\overline{\Delta}_j) & \text{o/w.} \end{cases} . \tag{12}$$

- Proceed as in Construction 8 to compute $\mathsf{Tables}_t = (\mathsf{evk}_t, \overline{\mathsf{evk}}_t, \mathbf{k}_{\mathrm{pub},t}, \overline{\mathbf{k}}_{\mathrm{pub},t})$ and labels $\{L_{x,t}^{(i)}\}_i$ from the t-th shares denoted as $\mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_j$. As a shorthand, we write

$$(\mathsf{Tables}_t, \{L_{x,t}^{(i)}\}_i) \leftarrow \mathsf{Comps}_{\mathbf{x}, C_{\mathrm{pub}}}(\mathsf{Keys}_t)$$

to mean the computations in this step, summarized here for reference.

$$\begin{aligned} &\mathsf{Comps}_{\mathbf{x}, C_{\mathrm{pub}}}(\mathsf{Keys} = \{\Delta_j, \overline{\Delta}_j\}_j) : \\ &\text{Denote } \boldsymbol{\Delta} = (\ldots, \Delta_j, \ldots)_j,\ \overline{\boldsymbol{\Delta}} = (\ldots, \overline{\Delta}_j, \ldots)_j, \\ &\text{Compute} \\ &(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta}), \qquad (\overline{\mathsf{sk}}, \overline{\mathsf{evk}}), \leftarrow \mathsf{KeyGen}(1^\lambda, \overline{\boldsymbol{\Delta}}), \\ &\quad \mathbf{k}_{\mathrm{pub}} \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C_{\mathrm{pub}}, \mathbf{id}), \qquad \overline{\mathbf{k}}_{\mathrm{pub}} \leftarrow \mathsf{EvalKey}(\overline{\mathsf{sk}}, \overline{C_{\mathrm{pub}}}, \overline{\mathbf{id}}) \\ &\quad \sigma_x^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \mathbf{id}[i]), \qquad \overline{\sigma}_x^{(i)} \leftarrow \mathsf{Auth}(\overline{\mathsf{sk}}, \mathbf{x}[i], \overline{\mathbf{id}}[i]), \\ &\text{Output } \mathsf{Tables} = (\mathsf{evk}, \overline{\mathsf{evk}}, \mathbf{k}_{\mathrm{pub}}, \overline{\mathbf{k}}_{\mathrm{pub}}), \text{ and } \{L_x^{(i)} = (\sigma_x^{(i)}, \overline{\sigma}_x^{(i)})\}_i. \end{aligned} \tag{13}$$

We further use the notation $\mathsf{Keys}_t[\mathbf{w}]$ to mean the subset:

$$\mathsf{Keys}_t[\mathbf{w}] = \{\Delta_{j,t} : \mathbf{w}[j] = 1\} \cup \{\overline{\Delta}_{j,t} : \mathbf{w}[j] = 0\}.$$

– Output the simulated garbled circuit $\widetilde{C} = (\widetilde{C_{\text{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \{\mathsf{Tables}_t\})$, and input labels $\{\widetilde{L}_x^{(i)} = (\ldots, L_{x,t}^{(i)}, \ldots)_t\}$ and $\{\widetilde{L}_y^{(i)}\}$.

We first argue that the function $\mathsf{Comps}_{\mathbf{x},C_{\text{pub}}}$ is a $1/(2\lambda)$-leaking computation with respect to the subset $\mathsf{Keys}_t[\overline{\mathbf{w}}]$.

Claim. For every polynomial $p(\lambda)$, and every efficient distinguisher $\mathcal{A}$, there exists a negligiable function negl such that for every $\lambda \in \mathbb{N}$, sequence of Boolean circuits $\{f_\lambda\}$ with $|f_\lambda| \leq p(\lambda)$, inputs $\{\mathbf{x}_\lambda\}$, and inputs $\{\mathsf{Keys}_\lambda\}$, the following holds.

$$\begin{aligned}&\left|\Pr[\mathcal{A}(\mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys})) = 1] - \Pr[\mathcal{A}(\mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}'))) = 1]\right| \\ &< 1/\lambda + \text{negl}(\lambda),\end{aligned}$$

where $\mathsf{Keys}'[\mathbf{w}] = \mathsf{Keys}[\mathbf{w}]$, and $\mathsf{Keys}'[\overline{\mathbf{w}}] = 0$, for $\mathbf{w} = f(\mathbf{x})$.

Proof (of Claim). We show a series of hybrids that transition from $\mathsf{Hyb}'_0 = \mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys})$ to $\mathsf{Hyb}'_3 = \mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}')$.

Hyb'_0 : The output of this hybrid is summarized in Equation 13.

Hyb'_1 : Instead of computing $\mathbf{k}_f$, $\overline{\mathbf{k}}_f$ as in Equation 13, Hyb'_1 simulates them as

$$\begin{aligned}&\sigma_{\mathbf{x}} := (\ldots, \sigma_x^{(i)}, \ldots)_i, && \overline{\sigma}_{\mathbf{x}} := (\ldots, \overline{\sigma}_x^{(i)}, \ldots)_i, \\ &\sigma_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{evk}, f, \mathbf{x}, \sigma_{\mathbf{x}}), && \overline{\sigma}_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\overline{\mathsf{evk}}, \overline{f}, \mathbf{x}, \overline{\sigma}_{\mathbf{x}}), \\ &\mathbf{k}_f[j] \leftarrow \begin{cases}\sigma_{\mathbf{w}}[j] - \Delta_j, \\ \sigma_{\mathbf{w}}[j],\end{cases} && \overline{\mathbf{k}}_f[j] \leftarrow \begin{cases}\overline{\sigma}_{\mathbf{w}}[j], & \text{if } \mathbf{w}[j] = 1, \\ \overline{\sigma}_{\mathbf{w}}[j] - \overline{\Delta}_j, & \text{o/w},\end{cases}\end{aligned} \tag{14}$$

where $\mathbf{w} = f(\mathbf{x})$. The $1/(2\lambda)$-correctness of aHMAC (Definition 13) ensures that the simulation is correct except with probability $\leq 1/(2\lambda)$. Hence

$$|\Pr[\mathcal{A}(\mathsf{Hyb}'_1) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}'_0) = 1| \leq 1/(2\lambda).$$

Hyb'_2 : Instead of computing evk, $\overline{\mathsf{evk}}$, $\sigma_{\mathbf{x}}$, and $\overline{\sigma}_{\mathbf{x}}$ as in Equation 13, Hyb'_2 simulates them using the simulator $\mathsf{aHMAC.Sim}$

$$(\mathsf{evk}, \sigma_{\mathbf{x}}) \leftarrow \mathsf{aHMAC.Sim}(1^\lambda), \quad (\overline{\mathsf{evk}}, \overline{\sigma}_{\mathbf{x}}) \leftarrow \mathsf{aHMAC.Sim}(1^\lambda).$$

The security of aHMAC ensures

$$|\Pr[\mathcal{A}(\mathsf{Hyb}'_2) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}'_1) = 1| \leq \text{negl}(\lambda).$$

Note that in Hyb'_2, the input keys $\{\Delta_j, \overline{\Delta}_j\}$ are only used for deriving $\mathbf{k}_f$, $\overline{\mathbf{k}}_f$ from the evaluated tags $\sigma_{\mathbf{w}}$, $\overline{\sigma}_{\mathbf{w}}$, as in Equation 14. In particular, Hyb'_2 is independent of $\mathsf{Keys}[\overline{\mathbf{w}}]$.

Hyb'_3 : The output of this hybrid is $\mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}')$, where $\mathsf{Keys}'[\mathbf{w}] = \mathsf{Keys}[\mathbf{w}]$, and $\mathsf{Keys}'[\overline{\mathbf{w}}] = 0$.

The same arguments from Hyb'_1, Hyb'_2, in reverse order, shows

$$|\Pr[\mathcal{A}(\mathsf{Hyb}'_3) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}'_2) = 1| \leq 1/(2\lambda) + \text{negl}(\lambda).$$

By a hybrid argument, we conclude that $|\Pr[\mathcal{A}(\mathsf{Hyb}'_3) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}'_0) = 1| \leq 1/\lambda + \text{negl}(\lambda)$, which proves the claim. □

We will then use the following lemma from [BGIK22], which is further based on a computational hardcore lemma from [MT10]. The lemma intuitively says that a δ-leaking distribution with respect to some secret m can be simulated by another distribution that completely leaks m with probability δ, and perfectly hides m with probability $1-\delta$.

Lemma 10 (Simulating Leaky Functions [BGIK22]). *Let* Leaky *be an efficiently computable randomized function with domain* $\mathcal{M}(\lambda)$*, and* $\delta = \delta(\lambda)$ *be a bound such that for every sequence of inputs* $\{m_\lambda\}, \{m'_\lambda\}$*, every polynomial distinguisher* $\mathcal{A}$*, it holds for sufficiently large* λ *that*

$$\left|\Pr[\mathcal{A}(1^\lambda, \mathsf{Leaky}(m_\lambda)) = 1] - \Pr[\mathcal{A}(1^\lambda, \mathsf{Leaky}(m'_\lambda)) = 1]\right| < \delta(\lambda).$$

Then, there exists randomized functions:

- $\mathsf{Erase}_\delta : \mathcal{M} \to \mathcal{M} \cup \{\bot\}$ *such that for every* $m \in \mathcal{M}$*,*

$$\Pr[\mathsf{Erase}_\delta(m) = m] \le \delta, \quad \Pr[\mathsf{Erase}_\delta(m) = \bot] = 1 - \Pr[\mathsf{Erase}_\delta(m) = m].$$

- $\mathsf{SimLeaky}$ *such that for every sequence* $\{m_\lambda\}$*,*

$$\{\mathsf{Leaky}(m_\lambda)\} \approx_c \{\mathsf{SimLeaky}(\mathsf{Erase}_\delta(m_\lambda))\},$$

  where Erase_δ *and* $\mathsf{SimLeaky}$ *depend on the same random coins.*

Let $\mathsf{Leaky}(\mathsf{Keys}[\overline{\mathbf{w}}]) := \mathsf{Comps}(\mathsf{Keys})$, it follows that there exists $\mathsf{Erase}_{1/\lambda}$, $\mathsf{SimLeaky}$ such that $\mathsf{Leaky}(\mathsf{Keys}[\overline{\mathbf{w}}]) \approx_c \mathsf{SimLeaky}(\mathsf{Erase}_{1/\lambda}(\mathsf{Keys}[\overline{\mathbf{w}}]))$.

We now show a series of hybrids that transitions from the real-world distribution in Definition 1 (Hyb_0) to the above simulated distribution (Hyb_6).

Hyb_0 : We summarize the real-world distribution of the garbled circuit $\widehat{C} = (\widehat{C_{\mathrm{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \{\mathsf{Tables}_t\})$ and labels $\{L_x^{(i)} = (\ldots, L_{x,t}^{(i)}, \ldots)\}$, $\{L_y^{(i)}\}$.

$$\begin{array}{l|l} L_y^{(i)} = K_y^{(i)}(\mathbf{y}[i]), \quad \widehat{C_{\mathrm{priv}}}, & \widehat{C_{\mathrm{priv}}}, \{K_w^{(j)}\}, \{K_y^{(i)}\} \leftarrow \mathsf{BG.Garb}(h, \{K_z^i\}), \\ \mathsf{ct}_j \leftarrow \mathsf{E.Enc}(\Delta_j, K_w^{(j)}(1)), & \\ \overline{\mathsf{ct}}_j \leftarrow \mathsf{E.Enc}(\overline{\Delta}_j, K_w^{(j)}(0)), & \Delta_j, \overline{\Delta}_j \leftarrow \{0,1\}^\lambda \end{array} \tag{15}$$

$$\begin{array}{l|l} (\{L_{x,t}^{(i)}\}, \mathsf{Tables}_t) & \{\Delta_{j,t}\} \leftarrow \mathsf{Share}(\Delta_j),\ \{\overline{\Delta}_{j,t}\} \leftarrow \mathsf{Share}(\overline{\Delta}_j) \\ \quad \leftarrow \mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}_t). & \mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_j \end{array} \tag{16}$$

Hyb_1 : Instead of computing $\{L_{x,t}^{(i)}\}, \mathsf{Tables}_t$ as in Equation 13, Hyb_1 simulates them using the $\mathsf{Erase}_{1/\lambda}$ and $\mathsf{SimLeaky}$ algorithms from Lemma 10:

$$(\{\widetilde{L}_{x,t}^{(i)}\}, \widetilde{\mathsf{Tables}_t}) \leftarrow \mathsf{SimLeaky}(\mathsf{Erase}_{1/\lambda}(\mathsf{Keys}_t[\overline{\mathbf{w}}])).$$

Lemma 10 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Proceeds as in Hyb_1, but aborts if there are $\ge \lambda$ instances (among 3λ) of $\mathsf{Erase}_{1/\lambda}$ that doesn't erase its input.

Since each instance of $\mathsf{Erase}_{1/\lambda}$ independently erases with probability $> 1 - 1/\lambda$, by Chernoff bound, abort happens with negligiable probability. Hence $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_1$.

Hyb_3 : Instead of computing the shares in as in Equation 13, Hyb_3 simulates them according to $\mathbf{w}$ as in Equation 12.

Note that the changed shares are exactly those in $\mathsf{Keys}_t[\overline{\mathbf{w}}]$, which are erased except for $\leq \lambda - 1$ indices t. The $(\lambda-1)$-privacy of secret sharing (Definition 19) ensures $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 : Change back to computing $(\{L^{(i)}_{x,t}\}, \mathsf{Tables}_t)$ from $\mathsf{Comps}_{\mathbf{x},C_{\text{pub}}}$ as in Equation 13, instead of using Erase_λ and $\mathsf{SimLeaky}$, as they are potentially inefficient. (The shares are still simulated as in Equation 12.)

Lemma 10 again ensures that $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_3$.

We draw attention to the keys Δ_j for $\mathbf{w}[j] = 0$, and $\overline{\Delta_j}$ for $\mathbf{w}[j] = 1$. Hyb_3 has changed from computing secret shares of those keys to secret shares of zeros. Hence they are not used in the current experiment except for encrypting the corresponding ciphertexts in $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$.

Hyb_5 : Instead of computing the ciphertexts $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$ as in Equation 15, simulate them as in Equation 11 (re-created here), where $L^{(j)}_w := K^{(j)}_w[\mathbf{w}[j]]$.

$$\mathsf{ct}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\Delta_j, L^{(j)}_w) \\ \mathsf{E.Enc}(\Delta_j, 0) \end{cases} \qquad \overline{\mathsf{ct}}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\overline{\Delta}_j, 0) & \text{if } \mathbf{w}[j] = 1 \\ \mathsf{E.Enc}(\overline{\Delta}_j, L^{(j)}_w) & \text{o/w} \ , \end{cases}$$

The semantic security of the encryption scheme E ensures that $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_4$.

Hyb_6 : Instead of computing $\widetilde{C_{\text{priv}}}$, $\{L^{(i)}_y\}$ and $\{L^j_w = K^{(j)}_w[\mathbf{w}[j]]\}$ as in Equation 15, simulate them using the simulator $\mathsf{BG.Sim}$ guaranteed by the security of Boolean garbling:

$$(\widetilde{C_{\text{priv}}}, \{\widetilde{L}^{(j)}_w\}, \{\widetilde{L}^{(i)}_y\}) \leftarrow \mathsf{BG.Sim}(1^\lambda, C_{\text{priv}}, \{L^{(i)}_z\}).$$

The security of Boolean garbling ensures $\mathsf{Hyb}_6 \approx \mathsf{Hyb}_5$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_5$, which proves the lemma. □

4.3 Implications: Succinct Secret Sharing, Garbling, and PSM

We first point out an immediate implication to succinct secret sharing for partite functions (See [ABI+23] for a formal definition). Such a secret sharing scheme has n pairs of shareholders (i.e., $2n$ in total), and an access structure define by a function $f : \{0,1\}^n \to \{0,1\}$ in the following way:

1. For all $\mathbf{x} \in \{0,1\}^n$ such that $f(\mathbf{x}) = 1$, the subset of n share holders, one from each i-th pair "selected" by $\mathbf{x}[i]$, can recover the secret.
2. A subset that contains two shareholders from any pair can recover the secret.
3. Subsets other than the above learn nothing about the secret.

Such a secret sharing scheme can be implemented by a partial garbling of the circuit $C(\mathbf{x}, s) = f(\mathbf{x}) \cdot s$, where s is the secret. Each of the n pairs of shareholders is assigned the two possible partial garbling input labels for $\mathbf{x}[i]$, and the garbled circuit $\widehat{C}$ can be released as public information or sent to all shareholders. [10]

Corollary 4 (Succinct Secret Sharing for Partite Functions). *There exists a succinct secret sharing for partite functions specified as Boolean circuits (of unbounded size), where the share sizes are* $\text{poly}(\lambda)$ *bits, assuming any for the assumptions from Theorem 9:*

[10] Technically we have only ensured condition 1, 3 of the access structure. We can additionally send an additive share of the secret to each pair of shareholders to ensure they can always jointly recover the secret.

1. *CP-DDH in either the NIDLS framework or prime-order groups;*
2. *the KDM-DCR assumption.*

There also exists a scheme for partite functions specified as bounded-depth (and unbounded size) Boolean circuits, where the share sizes are $\mathrm{poly}(\lambda)$ *bits, assuming any for the assumptions from Theorem 10:*

1. *DDH in either the NIDLS framework or prime-order groups;*
2. *The DCR assumption.*

Next, we sketch how to "upgrade" a partial garbling to a (fully private) standard garbling using a homomorphic encryption (HE) scheme, following the blueprint of [GKP+13]. To garble a program P, we define

$$P'(\mathbf{x}, \mathbf{y}) := \mathsf{HE.Dec}(\mathbf{y}, \mathsf{HE.Eval}(P, \mathbf{x})),$$

which is supposed to take HE ciphertexts as the public input $\mathbf{x}$, and a HE decryption key as the private input $\mathbf{y}$. A partial garbling of P' then implements a standard garbling of P. Intuitively, the partial garbling security ensures the HE decryption key (as the private input $\mathbf{y}$) is hidden, while the HE security ensures the actual input encrypted in HE ciphertexts (as the public input $\mathbf{x}$) are hidden.

Assuming a succinct partial garbling for circuits and a compact HE in the above construction, the resulting garbling scheme is also succinct (Definition 3). We further note that if the HE evaluation algorithm $\mathsf{HE.Eval}$ has bounded computation depth, then a succinct partial garbling for bounded-depth circuits suffices.

Definition 20 (Homomorphic Encryption Schemes (HE)). *A (secret-key) homomorphic encryption scheme for the class of programs* $\mathcal{P} = \{\mathcal{P}_\lambda\}$ *with Boolean inputs consists of four efficient algorithms.*

- $\mathsf{KeyGen}(1^\lambda)$ *outputs public parameters* pp *and a secret key* sk.
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}, x \in \{0,1\})$ *takes a secret key* sk *and a message* x. *It outputs a ciphertext* ct.
- $\mathsf{Eval}(\mathsf{pp}, P \in \mathcal{P}_\lambda, \{\mathsf{ct}_i\})$ *takes a program* $P : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$ *and* ℓ_x *ciphertexts. It outputs a evaluated ciphertext* ct^*.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*)$ *takes a (evaluated) ciphertext and outputs messages* $\mathbf{y} \in \{0,1\}^{\ell_y}$.

Correctness. *For every* $\lambda \in \mathbb{N}$, *program* $P \in \mathcal{P}_\lambda$ *with* ℓ_x *inputs, and input* $\mathbf{x} \in \{0,1\}^{\ell_x}$, *the following holds:*

$$\Pr\left[P(\mathbf{x}) = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*) \;\middle|\; \begin{array}{l} (\mathsf{sk}, \mathsf{pp}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \quad \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \mathbf{x}[i]) \\ \quad \mathsf{ct}^* \leftarrow \mathsf{Eval}(\mathsf{pp}, P, \{\mathsf{ct}_i\}) \end{array} \right] = 1.$$

Security. *The standard semantic security for secret-key encryption schemes should hold. (We omit writing it out here.)*

Compactness. *An evaluated ciphertext* ct^* *by* Eval *has bit-length* $\mathrm{poly}(\lambda, \ell_y)$ *independent of the program size, except the output length* ℓ_y.

Lemma 11 (HE Schemes). *There exist the following constructions:*

1. *[IP07, DGI+19] Assuming the DCR assumption or DDH in prime-order groups, for any polynomial $\ell(\lambda)$, there exists a compact homomorphic encryption scheme for the class of branching programs with bounded length by $\ell(\lambda)$ and unbounded size, i.e., $\mathcal{P}^{\ell} = \{\mathcal{P}^{\ell}_{\lambda}\}$ where $\mathcal{P}^{\ell}_{\lambda}$ consits of branching programs with length below $\ell(\lambda)$ and size below $2^{\mathrm{poly}(\lambda)}$.*
2. *[BGN05] Assuming the subgroup decision problem in bilinear groups of composite order, there exists a compact somewhat homomorphic encryption scheme for the class of quadratic polynomials (mod 2) of unbounded size, i.e., $\mathcal{Q} = \{\mathcal{Q}_{\lambda}\}$ where $\mathcal{Q}_{\lambda}$ consits of quadratic polynomials with below $2^{\mathrm{poly}(\lambda)}$ number of monomials.*

Furthermore, the computation depth of the Eval *algorithms in the above schemes are bounded by fixed polynomials* $\mathrm{poly}(\lambda)$*, independent of program sizes.*

Theorem 11 (Succinct Garbling for Bounded-Length BPs). *There exists a succinct garbling for bounded-length (and unbounded size) branching programs assuming (1) a succinct partial garbling scheme for bounded-depth circuits and (2) a compact homomorphic encryption scheme for bounded-length branching programs.*

Theorem 12 (Succinct Garbling for Quadratic Poly). *There exists a succinct garbling for quadratic polynomials (mod 2, and unbounded size) assuming (1) a succinct partial garbling scheme for bounded-depth circuits and (2) a compact homomorphic encryption scheme for quadratic polynomials.*

We point out truth tables as special cases of bounded-length branching programs: a truth table for ℓ_x inputs can be represented by a decision tree of depth ℓ_x, which is also a branching program of length ℓ_x. We therefore obtain succinct garbling for truth tables where the garbling size only depends polynomially on the input length.

As outlined in [FKN94], a garbling scheme implements a multi-party private simultaneous messages (PSM) protocol. we obtain the following corollary.

Corollary 5 (k-party Computational PSM for Truth Tables). *For any constant k,* [11] *any k-party function $f : [N]^k \to [N]$ has a computationally secure PSM protocol with* $\mathrm{poly}(\lambda, \log N)$ *communication and* $\mathrm{poly}(\lambda, N)$ *computation.*

Note that we have imposed composability of garbled circuits at the syntax level (Definition 1). So we can use the succinct garbling schemes for a program class $\mathcal{P} = \{\mathcal{P}_{\lambda}\}$ from Theorem 11 and 12 in an outer Boolean garbling scheme to handle general P-gates for any $P \in \mathcal{P}_{\lambda}$.

Corollary 6. *There exists a garbling scheme for circuits composed of general gates P that each implements any bounded-length (and unbounded size) branching program or any quadratic polynomial (mod 2, of unbounded size), where the garbling size is $\#wires \cdot \mathrm{poly}(\lambda)$, under the union of following assumptions:*

- *Any of the assumptions from Theorem 10;*
- *The DCR assumption, or DDH in prime-order groups;*
- *The subgroup decision problem in bilinear groups of composite order.*

[11] For super-constant k, the protocol will have communication $\mathrm{poly}(\lambda, \log(N^k))$, and computation $\mathrm{poly}(\lambda, N^k)$.

5 Application: 1-Key Selective CPRF for Circuits

The notion of constrained PRFs (CPRF) is first proposed (concurrently) in [BW13, KPTZ13, BGI14], where besides the usual pair of algorithms KeyGen and Eval for PRFs, there exists another pair $\mathsf{Constrain}$ and CEval. $\mathsf{Constrain}$ produces a constrained key sk_C with respect to a Boolean circuit C. CEval can use sk_C to evaluate the PRF on any point $\mathbf{x}$ such that $C(\mathbf{x}) = 0$.

The original definitions consider the setting where an adversary may adaptively obtain multiple constrained keys with respect to different circuits. This is referred to as multi-key CPRF. However, for circuit constraints (even low depth ones), multi-key CPRF is only known from heavy tools like indistinguishability obfuscation (iO) and functional encryption. Known non-iO based constructions [BV15, CC17, BTVW17, AMN+18, CVW18, PS18, CMPR23] only achieve the weaker definition, where the adversary obtains only a single selectively chosen constrained key. This is referred to as 1-key selective CPRF, and is also what we achieve in this work.

Definition 21 (Constrained Pseudorandom Functions). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ be classes of Boolean circuits where circuits in C_λ have domain $\mathcal{X}_\lambda$ and range $\{0,1\}$. A constrained pseudorandom function (CPRF) with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_\lambda$, key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_\lambda$, and range $\mathcal{Z} = \{\mathcal{Z}_\lambda\}_\lambda$ supporting circuit constraints $\mathcal{C}$ consists of four efficient algorithms:*

- $\mathsf{KeyGen}(1^\lambda)$ *outputs public parameters* pp *and a master secret key* msk.
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x \in \mathcal{X})$ *takes as inputs the master secret key* msk *and an input x. It outputs an evaluation result $z_0 \in \mathcal{Z}$.*
- $\mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, C \in \mathcal{C})$ *takes as inputs the master secret key* msk *and a constraint circuit C. It outputs a constrained key* sk_C.
- $\mathsf{CEval}(\mathsf{pp}, \mathsf{sk}_C, x)$ *takes as inputs the constrained key* sk_C *and an input x. It outputs an evaluation result $z_1 \in \mathcal{Z}$.*

Correctness: *There exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$, any circuit $C \in \mathcal{C}_\lambda$, and input $x \in \mathcal{X}_\lambda$ such that $C(x) = 0$, the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x) & (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ = \mathsf{CEval}(\mathsf{pp}, \mathsf{sk}_C, x) & \mathsf{sk}_C \leftarrow \mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, C) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

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

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

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

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

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

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.*

As explained in the technical overview, our construction relies on an extended syntax of homomorphic secret sharing schemes (HSS), formalized in Definition 22. We recap the observation of [CMPR23] that common HSS schemes for restricted multiplication straight-line programs (RMS) satisfy this syntax.

In an RMS program P, all inputs $\mathbf{x}$ are first converted into memory (i.e. intermediate) wires. Additions are allowed between two memory wires, but multiplications are restricted to between a memory wire and an input. Note that the initial conversion can be implemented by multiplying input wires with a constant memory wire of 1.

The observation is that if one change the initial conversion to using a constant memory wire of some value w instead of 1, while keeping the rest of the evaluation unchanged, then the final result becomes $w \cdot P(\mathbf{x})$.

The observation becomes useful in the context of evaluating RMS programs using HSS, because in common schemes the share format of memory wires are much simpler than the share format of inputs. In particular, any subtractive share (over $\mathbb{Z}$) of $\boldsymbol{\Delta} \cdot w$ is a valid memory share of w, where $\boldsymbol{\Delta}$ is a secret vector in the HSS scheme.

Definition 22 (Extended Homomorphic Secret Sharing). *An extended homomorphic secret sharing scheme for a class of programs $\mathcal{P}$ (defined over a ring $\mathcal{R}$) with input space $\mathcal{I} \subseteq \mathcal{R}$ consists of three efficient algorithms:*

- $\mathsf{Setup}(1^\lambda)$ *outputs a public key* pk*, a pair of evaluation keys* $\mathsf{evk}_0, \mathsf{evk}_1$*, and an "extension secret" as an integer vector* $\boldsymbol{\Delta} \in [2^\lambda]^{\ell_d}$*.*
- $\mathsf{Input}(\mathsf{pk}, x \in \mathcal{I})$ *takes as inputs the public key* pk *and an input x. It outputs a pair of input shares I_0, I_1.*
- $\mathsf{ExtEval}(\beta \in \{0,1\}, \mathsf{evk}_\beta, \mathbf{I}_\beta, \boldsymbol{\Delta}_\beta, P \in \mathcal{P})$ *takes as inputs a party identity β, its evaluation key* evk_β *and input shares $\mathbf{I}_\beta$, its share of the extension secret $\boldsymbol{\Delta}_\beta$, and a program P. It outputs its share of the evaluation result $z_\beta \in \mathcal{R}$.*

Extended Evaluation Correctness: *For all polynomial $p(\lambda)$, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$, any program $P \in \mathcal{P}$ with n inputs and 1 output, and with size $|P| \leq p(\lambda)$, any inputs $\mathbf{x} \in \mathcal{I}^n$, any extension bit $w \in \{0,1\}$, and any share vector $\boldsymbol{\Delta}_0 \in \mathbb{Z}^{\ell_d}$, the following holds:*

$$\Pr\left[\begin{array}{l} z_1 - z_0 = w \cdot P(\mathbf{x}) \\ (\text{over } \mathcal{R}) \end{array} \middle| \begin{array}{r} (\mathsf{pk}, \mathsf{evk}_0, \mathsf{evk}_1, \boldsymbol{\Delta}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (I_0^{(i)}, I_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, \mathbf{x}[i]) \\ \mathbf{I}_\beta := (I_\beta^{(0)}, \ldots, I_\beta^{(\ell_x - 1)}) \\ \boldsymbol{\Delta}_1 := \boldsymbol{\Delta}_0 + \boldsymbol{\Delta} \cdot w \quad (\text{over } \mathbb{Z}) \\ z_\beta \leftarrow \mathsf{ExtEval}(b, \mathsf{evk}_\beta, \mathbf{I}_\beta, \boldsymbol{\Delta}_\beta, P) \end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Security: *For any efficient adversary $\mathcal{A}$, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$ and for all $\beta \in \{0,1\}$:*

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

where the experiment $\mathrm{Exp}_{\mathrm{HSS}}^{\mathcal{A},\beta}$ *is as follows:*

1. *Launch* $\mathcal{A}(1^\lambda)$ *and receive challenge inputs* $x_0, x_1 \in \mathcal{I}$.
2. *Run* $(\mathsf{pk}, \mathsf{evk}_0, \mathsf{evk}_1, \boldsymbol{\Delta}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(I_0, I_1) \leftarrow \mathsf{Input}(\mathsf{pk}, x_b)$. *Then send* $(\mathsf{pk}, \mathsf{evk}_\beta, I_\beta)$ *to* $\mathcal{A}$.
3. *In the end,* $\mathcal{A}$ *outputs a bit* b' *as the experiment result.*

Remark 12. From an extended HSS we can obtain a "normal" HSS by viewing each party's evaluation key evk_β together with a subtractive share of $\boldsymbol{\Delta}$ as its overall evaluation key: $\mathsf{evk}^*_\beta := (\mathsf{evk}_\beta, \boldsymbol{\Delta}_\beta)$. This corresponds to setting the extension bit $w = 1$. Hence the evaluated shares satisfy $z_1 - z_0 = w \cdot P(\mathbf{x}) = P(\mathbf{x})$.

Lemma 12 (Extended HSS). *There exists an extended HSS scheme for NC1 assuming either of the following:*

1. *[OSY21] The DCR assumption.*
2. *[ADOS22] DDH and the small exponent assumption (Definition 10) in the NIDLS framework.*

In this section, we construct a CPRF by composing a leveled aHMAC and an HSS with extended evaluations.

Theorem 13 (CPRF for Circuits). *For any polynomial* $\ell(\lambda)$*, let* $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ *be the class of Boolean circuits with sizes bounded by* $\ell(\lambda)$*:*

$$\mathcal{C}_\lambda := \left\{ C : \{0,1\}^\lambda \to \{0,1\} : |C| < \ell(\lambda) \right\}.$$

There exists a 1-key selective CPRF for $\mathcal{C}$ *assuming either of the following:*

1. *DDH and the small exponent assumption in the NIDLS framework.*
2. *The DCR assumption.*

Construction 9 (1-Key Selective CPRF for Circuits). Ingredients:

- A leveled aHMAC scheme aHMAC with negl-correctness error.
- An extended HSS scheme HSS for a class $\mathcal{P}$ (defined over a ring $\mathcal{R}$) with input space $\mathcal{I} \subseteq \mathcal{R}$.
- A PRF $\mathsf{F} : \mathcal{K} \times \{0,1\}^\lambda \to \mathcal{R}$ with evaluation in $\mathcal{P}$, and with a compatible key space $\mathcal{K} = \mathcal{I}^{\ell_k}$.

We construct a CPRF for the class of polynomial-sized circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ where

$$\mathcal{C}_\lambda := \left\{ C : \{0,1\}^\lambda \to \{0,1\} : |C| < \mathrm{poly}(\lambda) \right\}.$$

As shorthands, in the following we write $U_{\mathbf{x}}(\cdot) = U(\cdot, \mathbf{x})$, and $\mathsf{F}_{\mathbf{x}}(\cdot) = \mathsf{F}(\cdot, \mathbf{x})$, where U is a universal circuit such that $U_{\mathbf{x}}(C) = C(\mathbf{x})$ for all circuits $C \in \mathcal{C}_\lambda$.

$(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$:

Sample a PRF key $\mathbf{s} \leftarrow \mathcal{I}^{\ell_k}$, and generate HSS input shares $\mathbf{I}_0 = (\ldots, I_0^{(i)}, \ldots)$, and $\mathbf{I}_1 = (\ldots, I_1^{(i)}, \ldots)$ of this key:

$$(\mathsf{pk}, \mathsf{HSS.evk}_0, \mathsf{HSS.evk}_1, \boldsymbol{\Delta}) \leftarrow \mathsf{HSS.Setup}(1^\lambda),$$
$$\text{for } i \in [\ell_k]: \quad (I_0^{(i)}, I_1^{(i)}) \leftarrow \mathsf{HSS.Input}(\mathsf{pk}, \mathbf{s}[i]).$$

Next generate aHMAC keys $\mathsf{sk}, \mathsf{aHMAC.evk}$ w.r.t. the extension secret $\boldsymbol{\Delta}$:

$$(\mathsf{sk}, \mathsf{aHMAC.evk}) \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, 1^{\mathrm{Depth}(U)}, \boldsymbol{\Delta}).$$

Output $\mathsf{pp} = \mathsf{pk}$ and $\mathsf{msk} = (\{I_\beta, \mathsf{HSS.evk}_\beta\}, \mathsf{sk}, \mathsf{aHMAC.evk})$.

$z_0 \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, \mathbf{x})$:

Parse I_0, $\mathsf{HSS.evk}_0$, and $\{\mathsf{sk}_j\}$ from msk.

Deterministically derive distinct **id** associated with inputs to $U_{\mathbf{x}}$, (i.e. the constraint circuit C,) and derive the zero-share $\boldsymbol{\Delta}_0$ of the extension secret:

$$\boldsymbol{\Delta}_0 := \mathbf{k}_U \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{sk}, U_{\mathbf{x}}, \mathbf{id}),^{12}$$

Next run extended HSS evaluation on the zero-shares of the input I_0 and the extension secret $\boldsymbol{\Delta}_0$:

$$z_0 \leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{evk}_0, \mathbf{I}_0, \boldsymbol{\Delta}_0, \mathsf{F}_{\mathbf{x}}).$$

$\mathsf{sk}_C \leftarrow \mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, C)$:

Parse I_1, $\mathsf{HSS.evk}_1$, and $\mathsf{sk}, \mathsf{aHMAC.evk}$ from msk.

Deterministically derive distinct **id** associated to the inputs to $U_{\mathbf{x}}$, and derive tags $\sigma_C := (\ldots, \sigma_C^{(i)}, \ldots)$ for C (viewed as a string):

$$\text{for } i \in \mathsf{bitLen}(C) \quad \sigma^{(i)} \leftarrow \mathsf{aHMAC.Auth}(\mathsf{sk}, C[i], \mathbf{id}[i]),$$

Output $\mathsf{sk}_C = (C, I_1, \mathsf{HSS.evk}_1, \sigma_C, \mathsf{aHMAC.evk})$.

$z_1 \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{sk}_C, \mathbf{x})$:

Parse C, I_1, $\mathsf{HSS.evk}_1$, σ_C, and $\mathsf{aHMAC.evk}$ from sk_C.

Derive the one-share $\boldsymbol{\Delta}_1$ of the extension secret:

$$\boldsymbol{\Delta}_1 := \sigma_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{evk}, U_{\mathbf{x}}, C, \sigma_C).$$

Note that aHMAC correctness ensures $\boldsymbol{\Delta}_1 = \boldsymbol{\Delta} \cdot C(\mathbf{x}) + \boldsymbol{\Delta}_0$ over $\mathbb{Z}$.

Next run extended HSS evaluation on the one-shares of the input I_1 and the extension secret $\boldsymbol{\Delta}_1$:

$$z_1 \leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{evk}_1, \mathbf{I}_1, \boldsymbol{\Delta}_1, \mathsf{F}_{\mathbf{x}}).$$

Note that extended HSS correctness ensures $z_1 = z_0 + C(\mathbf{x}) \cdot \mathsf{F}(\mathbf{s}, \mathbf{x})$.

Correctness: As noted in the construction, the evaluation results z_0, z_1 from Eval and CEval satisfy $z_1 = z_0 + C(\mathbf{x}) \cdot \mathsf{F}(\mathbf{s}, \mathbf{x})$, where C is the constraint circuit. When $C(\mathbf{x}) = 0$, we have $z_1 = z_0$ as desired.

Security: We state and prove the following security lemma.

Lemma 13. *Construction 9 is a 1-key selectively secure CPRF scheme.*

Proof. We show a series of hybrid experiments that transitions from the experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^{\mathcal{A},0}_{\mathrm{CPRF}}$ to $\mathsf{Hyb}_5 = \mathrm{Exp}^{\mathcal{A},1}_{\mathrm{CPRF}}$ as defined in Definition 21.

Hyb_0 : For reference, we summarize the adversary $\mathcal{A}$'s view, w.r.t a challenge circuit C in this experiment:

[12] The syntax of aHMAC requires an ℓ_d output circuit. We implicitly duplicate the one-bit output of $U_{\mathbf{x}}$ to ℓ_d bits here, and also in the construction of CEval.

– In the beginning, $\mathcal{A}$ receives public parameters $\mathsf{pp} = \mathsf{HSS.pk}$ and a constrained key sk_C the boxed terms in the following:

$$\begin{aligned}
&(\boxed{\mathsf{HSS.pk}}, \mathsf{HSS.evk}_0, \boxed{\mathsf{HSS.evk}_1}, \boldsymbol{\Delta}) \leftarrow \mathsf{HSS.Setup}(1^\lambda), \\
&\mathbf{s} \leftarrow \mathcal{I}^{\ell_k}, \quad (I_0^{(i)}, \boxed{I_1^{(i)}}) \leftarrow \mathsf{HSS.Input}(\mathsf{HSS.pk}, \mathbf{s}[i]),
\end{aligned} \tag{17}$$

$$\begin{aligned}
&(\mathsf{aHMAC.sk}, \boxed{\mathsf{aHMAC.evk}}) \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \boldsymbol{\Delta}), \\
&\boxed{\sigma^{(i)}} \leftarrow \mathsf{aHMAC.Auth}(\mathsf{aHMAC.sk}, C[i], \mathbf{id}[i]),
\end{aligned} \tag{18}$$

– For any number of (adaptively chosen) queries $\mathbf{x} \in \{0,1\}^\lambda$, $\mathcal{A}$ receives evaluations

$$\begin{aligned}
\mathbf{I}_0 &= (\ldots, I_0^{(i)}, \ldots), \\
\boldsymbol{\Delta}_0 = \mathbf{k}_U &\leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{aHMAC.sk}, U_\mathbf{x}, \mathbf{id}), \\
\boxed{z_0} &\leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{HSS.evk}_0, \mathbf{I}_0, \boldsymbol{\Delta}_0, \mathsf{F}_\mathbf{x}).
\end{aligned} \tag{19}$$

One of the query, $\mathbf{x}^*$ (with evaluation z_0^*) satisfying $C(\mathbf{x}) \neq 0$ is called the challenge query.

Hyb_1 : Instead of computing the evaluations to queries $\mathbf{x}$, including the challenge query, as in Equation 19, Hyb_1 simulate them as

$$\begin{aligned}
\mathbf{I}_1 &:= (\ldots, I_1^{(i)}, \ldots), \quad \sigma_C := (\ldots, \sigma^{(i)}, \ldots), \\
\boldsymbol{\Delta}_1 = \sigma_\mathbf{w} &\leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{aHMAC.evk}, U_\mathbf{x}, C, \sigma_C), \\
z_1 &\leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{HSS.evk}_1, \mathbf{I}_1, \boldsymbol{\Delta}_1, \mathsf{F}_\mathbf{x}), \\
\boxed{z_0} &\leftarrow z_1 - C(\mathbf{x}) \cdot \mathsf{F}(\mathbf{s}, \mathbf{x}).
\end{aligned}$$

The correctness of our Construction ensures that the above is an equivalent way of computing z_0, except with a negligible error probability. Hence $|\Pr[\mathcal{A}(\mathsf{Hyb}_1) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1| \leq \mathrm{negl}(\lambda)$.

Note that in Hyb_1, the evaluations are derived from the aHMAC tags σ_C and $\mathsf{aHMAC.evk}$, without depending on $\mathsf{aHMAC.sk}$ anymore.

Hyb_2 : Instead of computing σ_C and $\mathsf{aHMAC.evk}$ as in Equation 18, Hyb_2 simulates them using the simulator $\mathsf{aHMAC.Sim}$

$$(\boxed{\mathsf{evk}, \sigma_C}) \leftarrow \mathsf{aHMAC.Sim}(1^\lambda, 1^{\mathrm{Depth}(U)}).$$

aHMAC security ensures $|\Pr[\mathcal{A}(\mathsf{Hyb}_2) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_1) = 1| \leq \mathrm{negl}(\lambda)$.

Note that in Hyb_2, the share of extension secret $\boldsymbol{\Delta}_1$ is derived from the simulated aHMAC tags σ_C and evk, without depending on the actual secret $\boldsymbol{\Delta}$ anymore.

Hyb_3 : Instead of computing the HSS shares $I_1^{(i)}$ according to the PRF secret key $\mathbf{s}$ as in Equation 17, Hyb_3 simulate them as

$$(I_0^{(i)}, \boxed{I_1^{(i)}}) \leftarrow \mathsf{HSS.Input}(\mathsf{HSS.pk}, 0).$$

The security of HSS ensures $|\Pr[\mathcal{A}(\mathsf{Hyb}_3) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_2) = 1| \leq \mathrm{negl}(\lambda)$.

Note that in Hyb_3, the PRF secret key $\mathbf{s}$ is only used for evaluating $F(\mathbf{s}, \mathbf{x})$, and nowhere else.

Hyb_4 : Instead of answering the challenge query $\mathbf{x}^*$, satisfying $C(\mathbf{x}) = 1$, as $z_0^* \leftarrow z_1 - 1 \cdot \mathsf{F}(\mathbf{s}, \mathbf{x})$, Hyb_4 simulates it as

$$z_0^* \leftarrow z_1 - \mathsf{Uniform}(\mathcal{Z}_\lambda) \equiv \mathsf{Uniform}(\mathcal{Z}_\lambda),$$

where $\mathcal{Z}_\lambda$ is the output space of the CPRF and F.

PRF security (of F) ensures $|\Pr[\mathcal{A}(\mathsf{Hyb}_4) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_3) = 1| \leq \mathrm{negl}(\lambda)$.

Hyb_5 : This is the experiment $\mathrm{Exp}^{1,\lambda}_{\mathrm{CPRF}}$. The same arguments from Hyb_1 to Hyb_4, in reverse order, shows $|\Pr[\mathcal{A}(\mathsf{Hyb}_5) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_4) = 1| \leq \mathrm{negl}(\lambda)$.

By a hybrid argument, we conclude that $|\Pr[\mathcal{A}(\mathsf{Hyb}_5) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1| \leq \mathrm{negl}(\lambda)|$ which proves the lemma. □

Acknowledgments. Y. Ishai was supported by ISF grants 2774/20 and 3527/24, BSF grant 2022370, and ISF-NSFC grant 3127/23. H. Lin and H. Li were supported by NSF grant CNS-2026774, and a Simons Collaboration on the Theory of Algorithmic Fairness.

Bibliography

[AARV17] Benny Applebaum, Barak Arkis, Pavel Raykov, and Prashant Nalini Vasudevan. Conditional disclosure of secrets: Amplification, closure, amortization, lower-bounds, and separations. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 727–757. Springer, Cham, August 2017.

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09International Conference on Applied Cryptography and Network Security*, volume 5536 of *LNCS*, pages 292–305. Springer, Berlin, Heidelberg, June 2009.

[ABF+19] Benny Applebaum, Amos Beimel, Oriol Farràs, Oded Nir, and Naty Peter. Secret-sharing schemes for general and uniform access structures. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 441–471. Springer, Cham, May 2019.

[ABI+23] Benny Applebaum, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tianren Liu, and Vinod Vaikuntanathan. Succinct computational secret sharing. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1553–1566. ACM Press, June 2023.

[ABNP20] Benny Applebaum, Amos Beimel, Oded Nir, and Naty Peter. Better secret sharing via robust conditional disclosure of secrets. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd ACM STOC*, pages 280–293. ACM Press, June 2020.

[ACG24] Abtin Afshar, Jiaqi Cheng, and Rishab Goyal. Leveled fully-homomorphic signatures from batch arguments. Cryptology ePrint Archive, Report 2024/931, 2024.

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Cham, August 2022.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Berlin, Heidelberg, August 2013.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 119–135. Springer, Berlin, Heidelberg, May 2001.

[AL19] Leonard Assouline and Tianren Liu. Multi-party PSM, revisited. Cryptology ePrint Archive, Report 2019/657, 2019.

[AMN+18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Cham, August 2018.

[AN21] Benny Applebaum and Oded Nir. Upslices, downslices, and secret-sharing with complexity of 1.5^n. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 627–655, Virtual Event, August 2021. Springer, Cham.

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 557–577. Springer, Berlin, Heidelberg, May 2014.

[BCG+18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, Berlin, Heidelberg, May 2011.

[BFR13] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 863–874. ACM Press, November 2013.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer, 2014.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Berlin, Heidelberg, March 2014.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Berlin, Heidelberg, August 2016.

[BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 121–151. Springer, Cham, August 2022.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, Berlin, Heidelberg, February 2005.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, Berlin, Heidelberg, August 2008.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[BIKK14] Amos Beimel, Yuval Ishai, Ranjit Kumaresan, and Eyal Kushilevitz. On the cryptographic complexity of the worst functions. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 317–342. Springer, Berlin, Heidelberg, February 2014.

[BKN18] Amos Beimel, Eyal Kushilevitz, and Pnina Nissim. The complexity of multiparty PSM protocols and related models. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 287–318. Springer, Cham, April / May 2018.

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Cham, May 2019.

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 494–524. Springer, Berlin, Heidelberg, March 2017.

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

[BRS03] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, Berlin, Heidelberg, August 2003.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Cham, November 2017.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Berlin, Heidelberg, March 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Berlin, Heidelberg, December 2013.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 446–476. Springer, Cham, April / May 2017.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, Berlin, Heidelberg, May 2013.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 538–555. Springer, Berlin, Heidelberg, March 2014.

[CFOR12] Alfonso Cevallos, Serge Fehr, Rafail Ostrovsky, and Yuval Rabani. Unconditionally-secure robust secret sharing with compact shares. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 195–208. Springer, Berlin, Heidelberg, April 2012.

[CFT22] Dario Catalano, Dario Fiore, and Ida Tucker. Additive-homomorphic functional commitments and applications to homomorphic signatures. In Shweta Agrawal and

Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 159–188. Springer, Cham, December 2022.

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 194–224. Springer, Cham, April 2023.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 577–607. Springer, Cham, August 2018.

[DGI+19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Cham, August 2019.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Berlin, Heidelberg, February 2001.

[DKK18] Itai Dinur, Nathan Keller, and Ohad Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 213–242. Springer, Cham, August 2018.

[DKN+20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 559–589. Springer, Cham, August 2020.

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017*, volume 10592 of *LNCS*, pages 381–399. Springer, Cham, October 2017.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, Berlin, Heidelberg, April 2015.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984.

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 567–578. ACM Press, October 2015.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 803–815. Springer, Berlin, Heidelberg, July 2005.

[GU24] Romain Gay and Bogdan Ursu. On instantiating unleveled fully-homomorphic signatures from falsifiable assumptions. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part I*, volume 14601 of *LNCS*, pages 74–104. Springer, Cham, April 2024.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Berlin, Heidelberg, August 2015.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Berlin, Heidelberg, December 2013.

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

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 763–792. Springer, Cham, August 2020.

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

[HKKW19] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 357–376. Springer, Cham, February 2019.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th FOCS*, pages 415–434. IEEE Computer Society Press, October 2023.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings*, pages 174–184. IEEE Computer Society, 1997.

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

[ILL24] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct partial garbling from groups and applications. Cryptology ePrint Archive, Paper 2024/2073, 2024.

[ILL25] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. Cryptology ePrint Archive, Paper 2025/442, 2025.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 575–594. Springer, Berlin, Heidelberg, February 2007.

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

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1545–1552. ACM Press, June 2023.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015.

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

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Berlin, Heidelberg, July 2008.

[Lip05] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier López, Robert H. Deng, and Feng Bao, editors, *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2005.

[LV18] Tianren Liu and Vinod Vaikuntanathan. Breaking the circuit-size barrier in secret sharing. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 699–708. ACM Press, June 2018.

[LVW18] Tianren Liu, Vinod Vaikuntanathan, and Hoeteck Wee. Towards breaking the exponential barrier for general secret sharing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 567–596. Springer, Cham, April / May 2018.

[LWYY24] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. Garbled circuits with 1 bit per gate. Cryptology ePrint Archive, Paper 2024/1988, 2024.

[MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part IV*, volume 15367 of *Lecture Notes in Computer Science*, pages 71–97. Springer, 2024.

[MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. Cryptology ePrint Archive, Paper 2025/245, 2025.

[MT10] Ueli M. Maurer and Stefano Tessaro. A hardcore lemma for computational indistinguishability: Security amplification for arbitrarily weak PRGs with optimal stretch. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 237–254. Springer, Berlin, Heidelberg, February 2010.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.

[OS07] Rafail Ostrovsky and William E. Skeith III. A survey of single-database private information retrieval: Techniques and applications (invited talk). In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 393–411. Springer, Berlin, Heidelberg, April 2007.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Cham, October 2021.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Berlin, Heidelberg, May 1999.

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 675–701. Springer, Cham, March 2018.

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

[QWW21] Willy Quach, Brent Waters, and Daniel Wichs. Targeted lossy functions and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 424–453, Virtual Event, August 2021. Springer, Cham.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 73–85. ACM, 1989.

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

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021,*

Part III, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Cham.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Berlin, Heidelberg, May 1997.

[Ste98] Julien P. Stern. A new efficient all-or-nothing disclosure of secrets protocol. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 357–371. Springer, Berlin, Heidelberg, October 1998.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 616–637. Springer, Berlin, Heidelberg, February 2014.

[WW24] Hoeteck Wee and David J. Wu. Succinct functional commitments for circuits from k-sfLin. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 280–310. Springer, Cham, May 2024.

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

[Zha22] Mark Zhandry. To label, or not to label (in generic groups). In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 66–96. Springer, Cham, August 2022.

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

A aHMAC from Lattices

While the focus of this work are group-based constructions of algebraic homomorphic MACs, in this section we include a lattice based construction as well as a leveled variant for completeness.

The lattice construction was first presented in the subsequent work of [ILL25], based on a circular variant of the RingLWE assumption called circular-power-RingLWE (CP-RLWE), or alternatively a leveled variant based on power-RingLWE (P-RLWE), without the circular assumption. Here we present a different leveled construction based on the more standard RingLWE assumption with small secrets (i.e. with coefficients of polynomial magnitudes). This construction is analogous to the leveled group-based constructions in Section 3.4, and is inspired by the techniques from the recent work of [MORS25].

Theorem 14 (aHMAC from Lattices). *Let* $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ *be the public parameters specified in Construction 10. We have the following constructions:*

1. *Assuming CP-RLWE (Definition 24) with respect to* pp*, then there exists an aHMAC scheme for arbitrary Boolean circuits.*
2. *Assuming RLWE (Definition 23) with respect to* pp*, then there exists a leveled aHMAC scheme for bounded-depth Boolean circuits.*

In the above, an authentication tag (evaluated or not) costs $\mathrm{poly}(\lambda)$ *bits. In the non-leveled scheme, an* evk *costs* $\ell_z \cdot \mathrm{poly}(\lambda)$ *bits. And in the leveled scheme for bounded depth circuits by* D*, an* evk *costs* $(\ell_z + D) \cdot \mathrm{poly}(\lambda)$ *bits.*

Definition 23 (RingLWE). *We say the RingLWE assumption holds with respect to the ring* $\mathcal{R}(\lambda)$*, a modulus* $q(\lambda)$*, error and secret distributions* $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ *if the following holds for every polynomial* $m(\lambda)$*:*

$$\left\{ \begin{matrix} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}, \\ (\text{over } \mathcal{R}_{\mathrm{q}}) \end{matrix} \middle| \begin{matrix} s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}, \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ \mathbf{a} \leftarrow \mathcal{R}_q^m \end{matrix} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b} \leftarrow \mathcal{R}_q^m \right\}_\lambda$$

Definition 24 (Circular-Power-RingLWE). *We say the circular power RingLWE (CP-RLWE) assumption holds with respect to the ring* $\mathcal{R}(\lambda)$*, two modulus* $p(\lambda), q(\lambda)$ *such that* $q = p \cdot \alpha$*, error and secret distributions* $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ *if the following holds for every polynomial* $m(\lambda)$*:*

$$\left\{ \begin{matrix} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}_1,\ s^2 \cdot \mathbf{a} + \mathbf{e}_2 + s \cdot \alpha \\ (\text{over } \mathcal{R}_{\mathrm{q}}) \end{matrix} \middle| \begin{matrix} s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}_1, \mathbf{e}_2 \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ \mathbf{a} \leftarrow \mathcal{R}_q^m \end{matrix} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathcal{R}_q^m \right\}_\lambda$$

Construction 10 (aHMAC from CP-RLWE). The construction is with respect to the following RingLWE public parameters $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$:

- a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where $n \leq \mathrm{poly}(\lambda)$ is a power-of-two;
- two modulus $p > \lambda^{\omega(1)}$, and $q = p \cdot \alpha$, where $\alpha > p \cdot \lambda^{\omega(1)}$.
- error and secret distributions $\mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}} \subseteq \mathcal{R}$ with coefficients bounded by $\mathrm{poly}(\lambda)$.

Additionally, it relies on a PRF $\mathsf{F} : \mathcal{K} \times \{0,1\}^* \to \mathcal{R}_p$. In the following, for a ring element $r \in \mathcal{R}$ we abuse notation to write $\mathsf{BC}(r)$ to mean viewing the coefficients of r as an integer vector over $\mathbb{Z}^n$ and applying bit-composition $\mathsf{BC}(\cdot)$ it.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^D, \mathbf{\Delta})$: Sample a secret ring element $s \leftarrow \mathcal{D}_{\mathsf{sk}}$, and compute ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ as follows (we abuse notations to write $\mathbf{\Delta} \in \mathcal{R}_p^{\ell_z}$ as the vector of ring elements whose coefficients encode the input $\mathbf{\Delta} \in \{0,1\}^{\ell_z \times \lambda}$):

$$\begin{aligned} &a, \leftarrow \mathcal{R}_q,\ \mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}},\ \mathbf{e}' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_z}, \\ &\mathsf{ct}_s := (a, \underbrace{s \cdot a + e_1}_{b},\ \underbrace{s^2 \cdot a + e_2 - s \cdot \alpha}_{c}),\ \mathsf{ct}_\Delta := (\mathbf{a}', \underbrace{s \cdot \mathbf{a}' + \mathbf{e}' - \mathbf{\Delta} \cdot \alpha}_{\mathbf{b}'}). \end{aligned} \tag{20}$$

Finally, sample PRF keys $\mathsf{key}_1, \mathsf{key}_2 \leftarrow \mathcal{K}$. Output $\mathsf{sk} = (\mathsf{pp}, a, \mathbf{a}, s, \mathsf{key}_1, \mathsf{key}_2)$, and $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$.

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$: Parse the secret element s and the (secret) PRF key key_2 from sk. Then compute and output an authentication tag (technically viewing the coefficients as an integer vector over $\mathbb{Z}^n$).

$$\sigma_x = s \cdot x + \mathsf{F}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathcal{R}.$$

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$: Parse pp, ciphertexts $\mathsf{ct}_s = \{a, b, c\}$, $\mathsf{ct}_\Delta = \{\mathbf{a}', \mathbf{b}'\}$, and a PRF key key_1 from evk.

1. Assign the tags $\sigma_{\mathbf{x}}$ to corresponding input wires of C', and then a tag $\sigma^{(w)}$ to every output wire w of some gate in C' (with input wires w_1, w_2 and values x_1, x_2) following the topological order:

– For Add gates, set $\sigma^{(w)} := \sigma^{(w_1)} + \sigma^{(w_2)}$ over $\mathcal{R}$.
– For Mult gates, compute the output tag $\sigma^{(w)}$ as follows:

$$d^{(w)} := a \cdot \sigma^{(w_1)} \cdot \sigma^{(w_2)} - b \cdot (\sigma^{(w_1)} \cdot x_2 + \sigma^{(w_2)} \cdot x_1) + c \cdot x_1 \cdot x_2 \text{ over } \mathcal{R}_q,$$
$$\sigma^{(w)} := \lfloor d/\alpha \rfloor + \mathsf{F}(\mathsf{key}_1, w) \text{ over } \mathcal{R}_p.$$

We note the following invariant: if the input tags have the form $\sigma^{(w_1)} = s \cdot x_1 + k^{(w_1)}$, and $\sigma^{(w_2)} = s \cdot x_2 + k^{(w_2)}$, over $\mathcal{R}$, then the computed tag also has the form $\sigma^{(w)} = s \cdot z + k^{(w)}$ over $\mathcal{R}$. For Add gates, the invariant is immediate, with $k^{(w)} := k^{(w_1)} + k^{(w_2)}$ over $\mathcal{R}$. For Mult gates, we note the following core identity:

$$\sigma^{(w_1)} \cdot \sigma^{(w_2)} - s \cdot \left(\sigma^{(w_1)} x_2 + \sigma^{(w_2)} x_1\right) + s^2 z = k^{(w_1)} \cdot k^{(w_2)} \quad \text{over } \mathcal{R}_q.$$

Plugging in the fact that $b = s \cdot a + e_1$, $c = s^2 \cdot a + e_2 - s \cdot \alpha$, we obtain

$$d^{(w)} = s \cdot z \cdot \alpha + a \cdot k^{(w_1)} \cdot k^{(w_2)} + \text{error, where}$$
$$\text{error } = e_1(\sigma^{(w_1)} x_2 + \sigma^{(w_2)} x_1) + e_2 z, \text{ and } \|\text{error}\|_\infty \leq p \cdot \text{poly}(\lambda) \ll \alpha.$$

We have shown that the error term from $d^{(w)}$ is much smaller than α. Hence the rounding step removes it, except with negligible probability. (See Lemma 1 in [BKS19].)

$$\sigma^{(w)} = \lfloor d^{(w)}/\alpha \rfloor + \mathsf{F}(\mathsf{key}_1, w) = sz + \underbrace{\lfloor a k^{(w_1)} k^{(w_2)}/\alpha \rfloor + \mathsf{F}(\mathsf{key}_1, w)}_{k^{(w)}} \text{ over } \mathcal{R}_p.$$

Shifting by the (pseudo-)random factor $\mathsf{F}(\mathsf{key}_1, w)$ ensures that $\sigma^{(w)} = sz + k^{(w)}$ holds over $\mathcal{R}$ except with negligible probability, as long as $\|sz\|_\infty \ll p$.

2. Compute the final output tags $\sigma_{\mathbf{z}} = (\ldots, \mathsf{BC}(\sigma'^{(o_j)}), \ldots)_{j \in [\ell_z]}$, where $\{o_j\}$ are output wires of C' (with values $\{z_j\}$):

$$d'^{(o_j)} := \mathbf{a}'[j] \cdot \sigma^{(o_j)} - \mathbf{b}'[j] \cdot z_j,$$
$$\sigma'^{(o_j)} = \lfloor d'^{(o_j)}/\alpha \rfloor + \mathsf{F}(\mathsf{key}_1, o_j) \text{ over } \mathcal{R}_p$$

Similarly, we note if the tags $\sigma^{(o_j)}$ have the form $\sigma^{(o_j)} = s \cdot z_j + k^{(o_j)}$, then we have

$$\sigma'^{(o_j)} = \mathbf{\Delta}[j] \cdot z_j + k'^{(o_j)} \text{ over } \mathcal{R},$$
$$\text{w/ } k'^{(o_j)} := \lfloor \mathbf{a}'[j] \cdot k^{(o_j)}/\alpha \rfloor + \mathsf{F}(\mathsf{key}_1, o_j) \text{ over } \mathcal{R}_p$$
$$\Rightarrow \sigma_{\mathbf{z}} = \mathbf{\Delta} \odot z_j + k_C \quad \text{over } \mathcal{R} \quad \text{w/ } k_C := \mathsf{BC}(k'^{(o_j)}),$$

where in the last line we abuse notations to write $\mathbf{\Delta}$ as a vector in $\mathbb{Z}^{\ell_z}$.

$\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$: Parse the PRF keys $\mathsf{key}_1, \mathsf{key}_2$ and ring elements $a, \mathbf{a}'$ from sk. Then compute MAC keys $k^{(w_j)}$ associated with each input wire w_j of C':

$$k^{(w_j)} = \mathsf{F}(\mathsf{key}_2, \mathbf{id}[j]).$$

1. Assign a MAC key to every output wire w of some gate in C' (with input wires w_1, w_2) following the topological order:
   - For Add gates, set $k^{(w)} := k^{(w_1)} + k^{(w_2)}$ over $\mathcal{R}$.
   - For Mult gates, compute the output MAC key $k^{(w)}$ as follows:

$$k^{(w)} = \lfloor a k^{(w_1)} k^{(w_2)} / \alpha \rfloor + \mathsf{F}(\mathsf{key}_1, w) \text{ over } \mathcal{R}_p.$$

As noted before, we have $\sigma^{(w)} = s \cdot z + k^{(w)}$ over $\mathcal{R}$.

2. Compute the final output MAC keys $\mathbf{k}_C = (\ldots, \mathsf{BC}(k'^{(o_j)}), \ldots)_{j \in \ell_z}$, where $\{o_j\}$ are the output wires of C':

$$k'^{(o_j)} = \lfloor \mathbf{a}'[j] \cdot k^{(o_j)} / \alpha \rfloor + \mathsf{F}(\mathsf{key}_1, o_j) \text{ over } \mathcal{R}_p.$$

As noted before, we have $\sigma_{\mathbf{z}} = \mathbf{\Delta} \odot \mathbf{z} + \mathbf{k}_C$ over $\mathcal{R}$ as desired.

Correctness, Efficiency, and Security. As in the group-based constructions, we have broken up and embedded correctness analysis as notes in the above. We note that the tags output by Auth and the $\mathsf{EvalTag}$ all have bounded coefficients by $O(2^\lambda)$. Hence they have bit-lengths bounded by $O(\lambda \cdot n) = \mathrm{poly}(\lambda)$, and satisfy succinctness. The evaluation key evk contains mainly the ciphertexts ct_s, ct_Δ, which are $O(\ell_z)$ ring elements. In total, evk has bit-length $\ell_z \cdot \mathrm{poly}(\lambda)$.

We state and prove the following security lemma.

Lemma 14. *Under CP-RLWE with respect to the public parameters $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ in Construction 10, then Construction 10 is a secure aHMAC scheme.*

Proof. The security of an aHMAC scheme (Definition 13) requires simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ to simulate an evaluation key evk and adaptively queried authentication tags σ.

- Sim_1 samples all components of the simulated $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$ at random. In more detail, it samples a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, and random ciphertexts $\mathsf{ct}_s = (a, b, c)$, and $\mathsf{ct}_\Delta = (\mathbf{a}', \mathbf{b}')$:

$$a, b, c \leftarrow \mathcal{R}_q, \quad \mathbf{a}', \mathbf{b}' \leftarrow \mathcal{R}_q^{\ell_z}.$$

- Sim_2 samples the authentication tag at random $\widetilde{\sigma} \leftarrow [2^\lambda]^n$.

We show a series of hybrids that transitions from the real-world experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^0_{\mathrm{priv}}$ in Definition 13 to the simulation-world experiment $\mathsf{Hyb}_3 = \mathrm{Exp}^1_{\mathrm{priv}}$.

Hyb_0 : We summarize the real-world distribution of the evaluation key $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}_s = (\mathbf{h}, \mathbf{h}_1, \mathbf{h}_2)$, and $\mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}_1)$, and of the authentication tag σ for some query (x, id).

$$\begin{aligned}
&\mathsf{key}_1 \leftarrow \mathcal{K}_1, \\
&a \leftarrow \mathcal{R}_q,\ b = s \cdot a + e_1,\ c = s^2 \cdot a + e_2 + s \cdot \alpha && \left| \begin{array}{l} s \leftarrow \mathcal{D}_{\mathsf{sk}}, e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}}, \\ \mathbf{e}' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_z}, \end{array} \right. \quad (21) \\
&\mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ \mathbf{b}' = s \cdot \mathbf{a}' + \mathbf{e}' - \mathbf{\Delta} \cdot \alpha, && \\
&\sigma = s \cdot x + \mathsf{F}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathcal{R} && |\mathsf{key}_2 \leftarrow \mathcal{K}_2, \quad (22)
\end{aligned}$$

Hyb_1 : Instead of computing each tag σ as in Equation 22, Hyb_1 simulates it as $\widetilde{\sigma} \leftarrow [2^\lambda]^{\ell_s}$. The PRF security of F_2 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Instead of computing $b, c, \mathbf{b}'$ as in Equation 21, sample them directly at random:

$$a \leftarrow \mathcal{R}_q,\ b \leftarrow \mathcal{R}_q,\ c \leftarrow \mathcal{R}_q$$
$$\mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ \mathbf{b}' \leftarrow \mathcal{R}_q^{\ell_z}.$$

CP-RLWE ensures that $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$, which proves the lemma. □

Construction 11 (Leveled aHMAC from RLWE). This construction relies on the same ingradients as Construction 10.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^D, \boldsymbol{\Delta})$: Compared to Construction 10, the only difference is that the ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ (Equation 20) are replaced with per-level ciphertexts $\mathsf{ct}^{(j)}$ for $j \in [D]$, and a final one ct_Δ computed as follows. First, sample *two* secret exponents per level

$$\forall j = 0, \ldots, D, \quad s_L^{(j)}, s_R^{(j)} \leftarrow \mathcal{D}_{\mathsf{sk}}.$$

Then compute the ciphertexts $\{\mathsf{ct}^{(j)}\}$ and ct_Δ.

$$\begin{aligned}
\forall j \in [D], \quad \mathbf{a}^{(j)} &\leftarrow \mathcal{R}_q^2,\ \mathbf{e}_{1,L}^{(j)}, \mathbf{e}_{1,R}^{(j)} \leftarrow \mathcal{D}_{\mathsf{err}}^2,\ \mathbf{e}_2^{(j)} \leftarrow \mathcal{R}_p \\
\mathsf{ct}^{(j)} &:= \big(\mathbf{a}^{(j)}, s_L^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_{1,L}^{(j)}, s_R^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_{1,R}^{(j)}, \\
&\qquad s_L^{(j)} \cdot s_R^{(j)}\mathbf{a} + \mathbf{e}_2^{(j)} + (s_L^{(j+1)}, s_R^{(j+1)}) \cdot \alpha\big), \\
\text{for } j = D, \quad \mathbf{a}' &\leftarrow \mathcal{R}_q^{\ell_z},\ \mathbf{e}' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_z}, \\
\mathsf{ct}_\Delta &:= (\mathbf{a}', s_L^{(D)} \cdot \mathbf{a}' + \mathbf{e}' + \boldsymbol{\Delta} \cdot \alpha).
\end{aligned}$$

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$: Compared to Construction 10, the only difference is that the global secret s now becomes the level-0 secrets:

$$\sigma_x = (s_L^{(0)}, s_R^{(0)}) \cdot x + \mathsf{F}^2(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathcal{R}^2.$$

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$: Compared to Construction 10, there are two overall differences:

- Each tag assigned to an intermediate wire w, of depth j and with value x, used to have the form $s \cdot x + k^{(w)}$ for a global secret s, but now will have the form $\mathbf{s}^{(j)} \cdot x + \mathbf{k}^{(w)}$ for a per-level secret vector $\mathbf{s}^{(j)} := (s_L^{(j)}, s_R^{(j)})$.
- Before evaluating a gate, an additional step is required to ensure the tags on both input wires have the same-level secret vector.

We first present evaluation procedures for Add and Mult gates assuming both input tags $\sigma^{(w_1)}, \sigma^{(w_2)}$ have the same level-j secret vector.

- For Add gates, set $\sigma^{(w)} := \sigma^{(w_1)} + \sigma^{(w_2)}$ over $\mathcal{R}^2$.

> Note that if the input tags have the form $\sigma^{(w_1)} = \mathbf{s}^{(j)} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \mathbf{s}^{(j)} \cdot x_2 + \mathbf{k}^{(w_2)}$ over $\mathcal{R}$, then the computed tag also has the form $\sigma^{(w)} = \mathbf{s}^{(j)} \cdot z + \mathbf{k}^{(w)}$ over $\mathcal{R}^2$, where $\mathbf{k}^{(w)} = \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$.

– For Mult gates, parse $\sigma^{(w_1)} = (\sigma_L^{(w_1)}, \sigma_R^{(w_1)})$, $\sigma^{(w_2)} = (\sigma_L^{(w_2)}, \sigma_R^{(w_2)})$, and $\mathsf{ct}^{(j)} = (\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)})$. Compute

$$\mathbf{d}^{(w)} := \mathbf{a}^{(j)} \cdot \sigma_L^{(w_1)} \cdot \sigma_R^{(w_2)} - \mathbf{b}_R^{(j)} \cdot \sigma_L^{(w_1)} \cdot x_2 - \mathbf{b}_L^{(j)} \cdot \sigma_R^{(w_2)} \cdot x_1 + \mathbf{c}^{(j)} \cdot x_1 \cdot x_2 \text{ over } \mathcal{R}_q^2,$$
$$\sigma^{(w)} := \lfloor \mathbf{d}^{(w)}/\alpha \rfloor + \mathsf{F}^2(\mathsf{key}_1, w) \text{ over } \mathcal{R}_p^2.$$

We show if the input tags have the form $\sigma^{(w_1)} = \mathbf{s}^{(j)} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \mathbf{s}^{(j)} \cdot x_2 + \mathbf{k}^{(w_2)}$ over $\mathcal{R}^2$, then the computed tag has the form $\sigma^{(w)} = \mathbf{s}^{(j+1)} \cdot z + \mathbf{k}^{(w)}$ over $\mathcal{R}^2$, with the level-$(j+1)$ secret vector. We rely on the following core identity:

$$\sigma_L^{(w_1)}\sigma_R^{(w_2)} - s_R^{(j)} \cdot \left(\sigma_L^{(w_1)} x_2\right) - s_L^{(j)} \cdot \left(\sigma_R^{(w_2)} x_1\right) + s_L^{(j)} s_R^{(j)} z = \mathbf{k}_L^{(w_1)} \mathbf{k}_R^{(w_2)} \quad \text{over } \mathcal{R}_q.$$

Plugging in $\mathbf{b}_L^{(j)} = s_L^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_{1,L}^{(j)}$, $\mathbf{b}_R^{(j)} = s_R^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_{1,R}^{(j)}$, $\mathbf{c} = s_L^{(j)} s_R^{(j)} \mathbf{a}^{(j)} + \mathbf{e}_2^{(j)} + \mathbf{s}^{(j+1)} \cdot \alpha$, we obtain

$$\mathbf{d}^{(w)} = \mathbf{s}^{(j+1)} \cdot z \cdot \alpha + \mathbf{a} \cdot \mathbf{k}_L^{(w_1)} \mathbf{k}_R^{(w_2)} + \text{error, where}$$
$$\text{error} = \mathbf{e}_{1,R}^{(j)} \sigma_L^{(w_1)} x_2 + \mathbf{e}_{1,L}^{(j)} \sigma_R^{(w_2)} x_1 + \mathbf{e}_2^{(j)} z \text{ and } \|\text{error}\|_\infty \leq p \cdot \text{poly}(\lambda) \ll \alpha,$$
$$\Rightarrow \lfloor \mathbf{d}^{(w)}/\alpha \rfloor + \mathsf{F}^2(\mathsf{key}_1, w) = \mathbf{s}^{(j+1)} \cdot z + \underbrace{\lfloor \mathbf{a} \cdot \mathbf{k}_L^{(w_1)} \mathbf{k}_R^{(w_2)}/\alpha \rfloor + \mathsf{F}^2(\mathsf{key}_1, w)}_{\mathbf{k}^{(w)}} \text{ over } \mathcal{R}_p.$$
$$\Rightarrow \sigma^{(w)} = \mathbf{s}^{(j+1)} \cdot z + \mathbf{k}^{(w)} \text{ over } \mathcal{R},$$

where the last two equalities hold except with negligible probability, by the same arguments as in Construction 10.

We can now transform any level-j tag to a level-$(j+1)$ tag of the same value by applying the described Mult procedure with another level-j tag of the constant value 1. We can obtain level-j tag of 1 for all levels, by starting from an arbitray input tag (of level-0) and squaring it j times.

$\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$: As in Construction 10, perform matching evaluations over MAC keys in the same order as $\mathsf{EvalTag}$. We present evaluation procedures for Add and Mult gates assuming the input MAC keys are $\mathbf{k}^{(w_1)}, \mathbf{k}^{(w_2)}$, and the output wire w has depth j.

– For Add gates, set $\mathbf{k}^{(w)} := \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$ over $\mathcal{R}^2$.

As noted in $\mathsf{EvalTag}$, the matching evaluated tag equals $\sigma^{(w)} = \mathbf{s}^{(j)} z + \mathbf{k}^{(w)}$ over $\mathcal{R}^2$.

– For Mult gates, parse $\mathbf{k}^{(w_1)} = (\mathbf{k}_L^{(w_1)}, \mathbf{k}_R^{(w_1)})$, and $\mathbf{k}^{(w_2)} = (\mathbf{k}_L^{(w_2)}, \mathbf{k}_R^{(w_2)})$. Read $\mathbf{a}^{(j)}$ from sk, and compute

$$\mathbf{k}^{(w)} := \lfloor \mathbf{a}^{(j)} \cdot \mathbf{k}_L^{(w_1)} \mathbf{k}_R^{(w_2)}/\alpha \rfloor + \mathsf{F}^2(\mathsf{key}_1, w) \text{ over } \mathcal{R}_p.$$

As noted in $\mathsf{EvalTag}$, the matching evaluated tag equals $\sigma^{(w)} = \mathbf{s}^{(j+1)}z + \mathbf{k}^{(w)}$ over $\mathcal{R}^2$.

Correctness, Efficiency, and Security. As before, we have broken up and embedded correctness analysis as notes in the above. Compared to Construction 10, the leveled construction has a larger evk consisting of per-level ciphertexts $\{\mathsf{ct}^{(j)}\}_{[D]}$ of $\mathrm{poly}(\lambda)$ bits each, and a final one ct_Δ of $\ell_z \cdot \mathrm{poly}(\lambda)$ bits. In total, the bit-length of evk is bounded by $(D + \ell_z) \cdot \mathrm{poly}(\lambda)$. Finally, we state and prove the following security lemma.

Lemma 15. *Under RingLWE with respect to the public parameters* $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ *in Construction 10, Construction 11, is a secure leveled aHMAC scheme.*

Proof. The security of an aHMAC scheme (Definition 13) requires a pair of simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ to simulate an evaluation key evk and adaptively queried authentication tags σ.

- Sim_1 samples all components of the simulated $\mathsf{evk} = (\mathsf{pp}, \{\mathsf{ct}^{(j)}\}_{[D]}, \mathsf{ct}_\Delta, \mathsf{key}_1)$ at random. In more detail, it samples a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}$, and random ciphertexts $\mathsf{ct}^{(j)} = (\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)})$, and $\mathsf{ct}_\Delta = (\mathbf{a}', \mathbf{b}')$:

$$\forall j \in [D],\ \mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)} \leftarrow \mathcal{R}_q^2, \quad \mathbf{a}', \mathbf{b}' \leftarrow \mathcal{R}_q^{\ell_z}.$$

- Sim_2 samples the authentication tag at random $\widetilde{\sigma} \leftarrow [p]^{2n}$.

We show a series of hybrids that transitions from the real-world experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^0_{\mathrm{priv}}$ in Definition 13 to the simulation-world experiment $\mathsf{Hyb}_3 = \mathrm{Exp}^1_{\mathrm{priv}}$.

Hyb_0 : We summarize the real-world distribution of the evaluation key $\mathsf{evk} = (\mathsf{pp}, \{\mathsf{ct}^{(j)}\}, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}^{(j)} = (\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)})$, and $\mathsf{ct}_\Delta = (\mathbf{a}', \mathbf{b}')$, and of the authentication tag σ for some query (x, id).

$$\forall j \in [D]: \begin{array}{ll} \mathsf{key}_1 \leftarrow \mathcal{K}, & \left| s_L^{(j)}, s_R^{(j)} \leftarrow \mathcal{D}_{\mathsf{sk}},\ \forall j = 0, \ldots, D, \right. \\ \mathbf{a}^{(j)} \leftarrow \mathcal{R}_q^2,\ \mathbf{b}_L^{(j)} = s_L^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_{1,L}^{(j)}, & \left| \mathbf{e}_{1,L}^{(j)}, \mathbf{e}_{2,L}^{(j)} \leftarrow \mathcal{D}_{\mathsf{err}}^2, \right. \\ \mathbf{b}_R^{(j)} = s_R^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_{1,R}^{(j)}, & \left| \mathbf{e}_2^{(j)} \leftarrow \mathcal{R}_p^2, \right. \quad (23) \\ \mathbf{c}^{(j)} = s_L^{(j)}s_R^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_2^{(j)} - \mathbf{s}^{(j+1)} \cdot \alpha, & \left| \mathbf{s}^{(j+1)} := (s_L^{(j+1)}, s_R^{(j+1)}), \right. \\ \mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ \mathbf{b}' = s_L^{(D)}\mathbf{a}' + \mathbf{e}' - \boldsymbol{\Delta} \cdot \alpha, & \left| \mathbf{e}' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_z}. \right. \quad (24) \\ \sigma = \mathbf{s}^{(0)} \cdot x + \mathsf{F}^2(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathcal{R} & |\mathsf{key}_2 \leftarrow \mathcal{K}, \quad (25) \end{array}$$

Hyb_1 : Instead of computing each tag σ as in Equation 25, Hyb_1 simulates it as $\widetilde{\sigma} \leftarrow [p]^{2n}$. The PRF security of F ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

$\mathsf{Hyb}_{2,0,0}$: Instead of computing $\mathbf{c}^{(0)}$ as in Equation 23, compute it in terms of $\mathbf{b}_R^{(0)}$:

$$\mathbf{c}^{(0)} = s_L^{(0)} \underbrace{(s_R^{(0)}\mathbf{a}^{(0)} + \mathbf{e}_{1,R}^{(0)})}_{\mathbf{b}_R^{(0)}} + \mathbf{e}'^{(0)}_2 + \mathbf{e}_2^{(0)} - \mathbf{s}^{(1)} \cdot \alpha \left| \mathbf{e}'^{(0)}_2 \leftarrow \mathcal{D}_{\mathsf{err}}^2,\ \mathbf{e}_2^{(0)} \leftarrow \mathcal{R}_p^2. \right.$$

By our setting, we have $\|s_L^{(0)}\mathbf{e}_{1,R}^{(0)}\|_\infty \ll p$ and $\|\mathbf{e}'^{(0)}_2\|_\infty \ll p$. Hence $\mathsf{Hyb}_{2,0,0} \approx \mathsf{Hyb}_1$.

$\mathsf{Hyb}_{2,0,1}$: Instead of computing $\mathbf{b}_L^{(0)}, \mathbf{b}_R^{(0)}$ as in Equation 23, sample them at random:

$$\left.\begin{aligned}&\mathbf{a}^{(0)}, \mathbf{b}_L^{(0)}, \mathbf{b}_R^{(0)} \leftarrow \mathcal{R}_q^2,\\ &\mathbf{c}^{(0)} = s_L^{(0)}\mathbf{b}_R^{(0)} + \mathbf{e}_2' + \mathbf{e}_2 - \mathbf{s}^{(1)} \cdot \alpha,\end{aligned}\right| \mathbf{e}_2' \leftarrow \mathcal{D}_{\mathsf{err}}^2,\ \mathbf{e}_2 \leftarrow \mathcal{R}_p^2.$$

By RingLWE, we have $\mathsf{Hyb}_{2,0,1} \approx_c \mathsf{Hyb}_{2,0,0}$.

$\mathsf{Hyb}_{2,0,2}$**:** Instead of computing $\mathbf{c}^{(0)}$ as the previous hybrid, directly sample it at random:

$$\mathbf{c}^{(0)} \leftarrow \mathcal{R}_q^2,$$

without depending on the secret vector $\mathbf{s}^{(1)}$. By RingLWE, we have $\mathsf{Hyb}_{2,0,2} \approx \mathsf{Hyb}_{2,0,1}$.

$\mathsf{Hyb}_{2,j}$: for $j = 1, \ldots D-1$, instead of computing $\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)}$ as in Equation 7, sample them at random:

$$\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)} \leftarrow \mathcal{R}_q^2.$$

By analogous arguments from $\mathsf{Hyb}_{2,0,0}$ and $\mathsf{Hyb}_{2,0,2}$, we have $\mathsf{Hyb}_{2,j} \approx_c \mathsf{Hyb}_{2,j-1}$.

Hyb_3 Instead of computing $\mathbf{a}', \mathbf{b}'$ as in Equation 8, sample them at random:

$$\mathbf{a}', \mathbf{b}' \leftarrow \mathcal{R}_q^{\ell_z}.$$

By RingLWE, we have $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_{2,D-1}$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$, which proves the lemma. □