Succinct Partial Garbling 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. A *garbling scheme* transforms a program (e.g., circuit) C 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, such that $(C, \hat{C}, \{k_{i,x_i}\})$ can be used to recover the output $z = C(x)$ while revealing nothing else about the input x. This can be naturally generalized to *partial garbling*, where part of the input is *public*, and a computation $z = C(x, y)$ is decomposed into a public part $C_{\text{pub}}(x)$, depending only on the public input x, and a private part $z = C_{\text{priv}}(C_{\text{pub}}(x), y)$ that also involves a private input y.

A key challenge in garbling is to achieve *succinctness*, where the size of the garbled program may grow only with the security parameter and (possibly) the output length, but not with the size of C. Prior work achieved this strong notion of succinctness using heavy tools such as indistinguishability obfuscation (iO) or a combination of fully homomorphic encryption and attribute-based encryption. In this work, we introduce new succinct garbling schemes based on variants of standard group-based assumptions. Our approach, being different from prior methods, offers a promising pathway towards practical succinct garbling. Specifically, we construct:

- A *succinct partial garbling scheme* for general circuits, where the garbled circuit size scales linearly with the private computation $|C_{\text{priv}}|$ and is independent of the public computation $|C_{\text{pub}}|$. This implies fully succinct *conditional disclosure of secrets* (CDS) protocols for circuits.
- *Succinct (fully hiding) garbling schemes* for simple types of programs, including truth tables, bounded-length branching programs (capturing decision trees and DFAs as special cases) and degree-2 polynomials, where the garbled program size is independent of the program size. This implies succinct *private simultaneous messages* (PSM) protocols for the same programs.

Our succinct partial garbling scheme can be based on a circular-security variant of the power-DDH assumption, which holds in the generic group model, or alternatively on the key-dependent message security of the Damgård-Jurik encryption. For bounded-depth circuits or the aforementioned simple programs, we avoid circular-security assumptions entirely.

At the heart of our technical approach is a new computational flavor of *algebraic homomorphic MAC* (aHMAC), for which we obtain group-based constructions building on techniques from the literature on homomorphic secret sharing. Beyond succinct garbling, we demonstrate the utility of aHMAC by constructing *constrained pseudorandom functions* (CPRFs) for general constraint circuits from group-based assumptions. Previous CPRF constructions were limited to NC^1 circuits or alternatively relied on lattices or iO.

Table of Contents

1 Introduction

Garbling schemes [94] play a critical role in modern cryptography, enabling non-interactive secure computation in a variety of applications. A garbling scheme can be viewed as a simple form of functional encryption, allowing a one-time computation on encrypted data while maintaining the strongest possible privacy guarantee.

More concretely, 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 [63] 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 [47], and conditional disclosure of secrets (CDS) [52, 5].

Yao's original garbling scheme for circuits [94] has been the subject of substantial optimization efforts [12, 80, 69, 85, 68, 58, 95, 89]. These works have reduced the size of garbled circuits to 1.5 ciphertexts per AND gate [89], while maintaining practical computational efficiency using only symmetric-key cryptography. However, these garbled circuits still grow linearly with the size of the original circuits, creating a communication bottleneck for large-scale computations.

Succinct Garbling. This bottleneck motivates the notion of *succinct garbling*, where the size of the garbled program $\hat{C}$ does not grow with the size of the original program C. Instead, $|\hat{C}|$ can depend (polynomially) only on the security parameter, and possibly also on second-order parameters such as the input and output length. Despite their promise, existing succinct garbling schemes rely on "high-end" primitives, such as indistinguishability obfuscation (iO) [72, 16] or combinations of fully homomorphic encryption (FHE) and attribute-based encryption (ABE) [54, 18, 61]. These approaches not only incur a high computational overhead but also are limited to specific cryptographic assumptions, such as (circular-secure) LWE [49, 29, 51] or combinations of multiple assumptions (e.g., LPN over large fields, local PRG, and DLin over bilinear groups) [64, 65, 88]. Even for the weaker notion of partial garbling with a constant-size private program, corresponding to CDS, succinct schemes without iO or lattices are only known for highly restricted program classes, such as truth tables [60, 59, 6] (see Section 1.2).

This raises a natural and compelling question:

Can we construct succinct garbling schemes for expressive classes of programs, without relying on iO or lattices?

Our Results in a Nutshell. In this work, we present a very different approach to succinct garbling, relying on group-based assumptions such as (circular-security variants of) the power-DDH assumption [55, 31, 7] or Paillier/Damgård-Jurik encryption [83, 41]. While not yet practical, our approach offers a new pathway toward concretely efficient succinct garbling by avoiding the reliance on iO or lattices. Instead, it builds on techniques from the rich literature on *homomorphic secret sharing* (HSS) (e.g., [23, 24, 1, 79]), which were based on a broader set of assumptions and were demonstrated to have a potential for concrete efficiency [24, 22, 27]. Our technical approach is inspired by the recent work of Meyer, Orlandi, Roy, and Scholl [79], which was the first to

apply HSS techniques in the context of garbling, but extends their ideas significantly. While [79] showed that HSS can be used to garble arithmetic circuits with a linear dependence on the circuit size, improving the hidden constant to 1, we demonstrate how to break this linear dependence and achieve full succinctness.

Specifically, we obtain the following group-based succinct garbling schemes:

- A *succinct partial garbling scheme* for general circuits, where the garbled size scales linearly with the private computation $|C_{\text{priv}}|$ and independently of the public computation $|C_{\text{pub}}|$. This implies succinct (computationally secure) multiparty CDS protocols for general circuits, where the message size does not grow with the circuit size.[3]
- *Succinct (fully hiding) garbling schemes* for simple but useful classes of programs, including truth tables, bounded-length branching programs (capturing DFAs and decision trees) and degree-two polynomials, where the garbled size is independent of the program size. This implies succinct *private simultaneous messages* (PSM) protocols [46] for the same classes.

All of the above schemes achieve an exponential improvement in succinctness compared to the state-of-the-art schemes based on group-related assumptions, without iO or lattices.

Our fully succinct garbling schemes are obtained by combining (group-based) somewhat homomorphic encryption with our partial garbling scheme. This replaces the ABE component in the previous approach from [54, 18] by the simpler partial garbling primitive.

At the heart of our partial garbling scheme is an algebraic homomorphic MAC (aHMAC) scheme, a refinement of fully homomorphic MACs [48] with the feature that a MAC of a message x has the commonly used algebraic form $\Delta \cdot x + K$. We believe that aHMAC is a versatile tool that can be of independent interest. In particular, we demonstrate its utility beyond succinct garbling by constructing *constrained pseudorandom functions* (CPRFs) for general constraint circuits. This result, which follows as a byproduct of our techniques, significantly extends the reach of group-based assumptions. Previously, CPRFs for general circuits were only achievable using iO or lattices, whereas group-based CPRFs were limited to NC^1 circuits [53, 9, 42, 40].

1.1 Our Results in More Detail

Our Assumptions. All of our results can be based on natural flavors of the Power Decisional Diffie Hellman (P-DDH) assumption (Definition 7), introduced in [55, 31, 7] and further used in [57, 70, 9, 11]. P-DDH postulates that for appropriately sampled group element g and exponents s and a, b sampled randomly from a range $[\ell]$, the triple (g, g^s, g^{s^2}) is indistinguishable from (g, g^a, g^b). This simple varaint of P-DDH is sufficient for most of our results, including succinct partial garbling and aHMAC for *bounded* depth circuits, CPRFs for *general circuits*, and succinct garbling for simple programs (additionally relying on other standard group-based assumptions).

To support circuits of an arbitrary depth (in succinct partial garbling and aHMAC), we need the following circular-security variant of this assumption. The Circular Power Decisional Diffie Hellman (CP-DDH) assumption (Definition 8) 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

[3] A multiparty CDS protocol [52] for predicate circuit C, where each party holds a public input bit x_i and the secret y is a single bit, can be viewed as a partial garbling of $C(x) \wedge y$. Here $C_{pub} = C$ and $C_{priv} = \wedge$. A succinct partial garbling can be obtained *generically* from any succinct multiparty CDS protocol and standard garbling, see Section 2.2.

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 P-DDH and CP-DDH assumptions 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, P-DDH and CP-DDH hold in the standard generic group model (GGM) [91]; see Section 3.3. In particular, our 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 10) [17, 20]. The semantic security of this cryptosystem relies on the Decisional Composite Residuosity (DCR) assumption [41]. KDM variants of this assumption are commonly used in prior Paillier-group based HSS works, for example [45, 81, 79].[4] We write KDM-DCR as a shorthand.

Note that the reliance of some of our results on circular-security assumptions aligns with the literature on FHE and lattice-based succinct garbling. For supporting arbitrary depth circuits, these constructions also rely on the circular-security variant of LWE, while standard LWE only enables HE and succinct garbling for bounded-depth circuits.

Succinct Partial Garbling. Based on the assumptions discussed above, we first obtain a succinct partial garbling scheme for general circuits.

Theorem 1 (Succinct Partial Garbling for Circuits, Informal; see Theorems 7, 8). *Assume both P-DDH and DDH*[5] *over* either *prime-order groups, Paillier/Damgård-Jurik 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 stronger CP-DDH assumption, or alternatively KDM-DCR, the garbled circuit size is reduced to $|C_{\text{priv}}| \cdot \text{poly}(\lambda)$.*

From Partial to Full Garbling. 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. The transformation is extremely simple, and follows the blueprint of constructing succinct garbled circuits from FHE and ABE [54, 18], 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.

[4] More concretely, our notion of KDM security assumes that encryptions (under sk) of sk^{-1} are secure. This is strictly weaker than [79], which assumes encryptions of any affine function of $\mathsf{sk}, \mathsf{sk}^{-1}$ are secure, and incomparable with [45, 81], which assume encryptions of bits of sk are secure.

[5] P-DDH implies DDH in prime-order groups, but this is not known in Damgård-Jurik groups or class groups. On the other hand, CP-DDH implies DDH in all groups.

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 [54, 18, 86]. 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). *Assume the same assumptions as in Theorem 1. Any homomorphic encryption scheme for a class of programs $\mathcal{P}$ can be transformed into a succinct garbling scheme for $\mathcal{P}$. The size of the garbled program is* $\mathrm{poly}(\lambda) \cdot m$ *if assuming circular security, or* $\mathrm{poly}(\lambda) \cdot (m + D_{\mathsf{Eval}})$ *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 [73, 92, 82, 74, 50, 44], can be viewed as a (compact) HE scheme for *truth-table* programs. This was generalized in [62, 44] 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.[6] 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 [19] 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 P-DDH and DDH over any of the groups in Theorem 1, 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* $\mathrm{poly}(\lambda) \cdot \#\mathit{wires}$*, where $\#wires$ is the number of wires in the circuit.*

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

Implication for PSM protocols. Succinct garbling can be directly applied to improve the state of the art on (computationally secure) *private simultaneous messages* (PSM) protocols [46]. In a PSM protocol, two or more parties who share common randomness wish to securely evaluate a function of their inputs by simultaneously sending messages to a referee. Our succinct garbling schemes for truth tables 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). *Under the assumptions in the first bullet of Corollary 1, 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.*

This yields exponential improvement in communication compared to previous information-theoretic PSM protocols whose complexity grows polynomially with N: $O(N^{1/2})$ for $k = 2$ [14] and $O_k(N^{(k-1)/2})$ for infinitely many k [15, 8]. No better PSM protocols were known under group-based assumptions.

PSM vs. CDS vs. generalized secret sharing. It is instructive to put the above PSM result in the context of related primitives. A k-party CDS protocol [52] is defined similarly to PSM, except that there is only one input bit hidden from the referee. Viewing CDS as an instance of partial garbling (see Footnote 3), our partial garbling result implies a stronger version of Corollary 3 for CDS that applies to *general circuits* (with computation that scales polynomially with the circuit size) rather than just applying to truth tables or other kinds of simple programs. Using the known relation between CDS and generalized secret sharing [52, 76], this implies fully succinct computational secret sharing for k-slice access structures[7] represented by *circuits*.

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 [6], which applies to *general* access structures represented by a truth table. One might a-priori expect the SCSS question to be easier because of 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 [6] 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 [18], 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.

Tool: Algebraic Homomorphic MACs. Our key technical tool is a new notion of an *algebraic* homomorphic MAC (aHMAC). A standard homomorphic MAC, introduced by Gennaro and Wichs [48] (following Agrawal and Boneh [3]), 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 tags to generate a

[7] In a k-slice access structure, coalitions of size k are either authorized or not, larger coalitions are authorized and smaller are unauthorized. For the simplest case of a *truth-table* representation of the access structure, fully succinct computational secret sharing can be based on one-way functions [6], and partially succinct schemes are known to exist even in the information-theoretic setting (see [76, 13] and references therein).

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. The key feature of homomorphic MAC is that the output tags are short, of size $\text{poly}(\lambda)$, 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). It has been shown that homomorphic MAC can be constructed from a variety of assumptions, including LWE and various group-based assumptions [33, 10, 34, 56, 4, 2]. See Section 1.2 for a survey of this line of work.

Our notion of aHMAC enhances traditional homomorphic MAC by requiring the output tags to have the most widely used algebraic form $\mathbf{\Delta}m + \mathbf{K}$, as in a standard information-theoretic MAC, where $\mathbf{\Delta}$ is a λ-bit global authentication key and K is a pseudorandom blinding term. In particular, in our scheme, 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 construct an algebraic homomorphic MAC for circuits, under the group-based assumptions stated in Theorem 1.

Theorem 2 (Algebraic Homomorphic MAC, Informal; see Theorem 5, 6). *Assume P-DDH and DDH over any of the groups from Theorem 1. Then there is an aHMAC (with output 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 stronger CP-DDH assumption, or alternatively KDM-DCR, the aHMAC can support general circuits with fixed $\text{poly}(\lambda)$-size evaluation key.*

Application: Constrained Pseudorandom Functions. Thanks to the algebraic form of the tags, our aHMAC scheme can be easily combined with other tools such as standard garbling and HSS. This is what we leverage in our construction of succinct partial garbling (combining with standard garbling). As another application, we show the usefulness of the aHMAC primitive for constructing a general *constrained pseudorandom function* (CPRF) [21, 67, 26]. Concretely, by combining an aHMAC with HSS schemes, and following the HSS-based blueprint of Couteau, Meyer, Passelègue and Riahinia [40], 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 11). *There is a 1-key, selectively-secure CPRF supporting general constraint circuits based on P-DDH, DDH, and small-exponent assumptions over Paillier/Damgård-Jurik groups or class groups.*

Previously, CPRF for general constraint circuits were only known under LWE [30, 32, 28, 36, 84] or iO, and constructions based on other assumptions were restricted to (low-depth) NC^1 constraints [53, 9, 42, 40].

On Our aHMAC Construction. Being our key technical tool, we provide here a brief overview of our aHMAC construction. See Section 2.1 for a detailed technical overview. Recall that a homomorphic secret sharing (HSS) scheme [23] is a 2-party analogue of FHE, allowing a local mapping from shares of an input x to *additive* shares of an output $f(x)$. Our construction builds heavily on techniques from the HSS literature. 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 by so-called "RMS programs" and *ii)* in standard group-based HSS schemes, *both* shares depend on the input. In aHMAC, and similarly in

garbling, 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 the computation f). Similarly, in partial garbling, the evaluator's computation can depend on public inputs, but not the garbler's.

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 garbler) is independent of x, y.

Our technique is inspired by the recent work of Meyer, Orlandi, Roy, and Scholl [79], which obtained an improved garbling scheme for arithmetic circuits over bounded integers assuming the KDM security of the Damgård-Jurik cryptosystem. To compare this with our aHMAC and partial garbling constructions, their garbled circuits hide the inputs, but have size $|C| \log B$ where B is an upper bound on the wire values. In contrast, our constructions reveal a public input, but have size independent of the public computation. The work of [79] already needs to overcome the RMS limitation of HSS. Essentially, they do so by designing 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. For each multiplication, $\log B$ bits of auxiliary information are needed, leading to the large size of the garbled circuit.

1.2 Related Works

Partial Garbling. The concept of partial garbling was first introduced by Ishai and Wee [63], extending traditional garbling schemes to settings where part of the input is public. Partial garbling enables computations to be decomposed into a public part, $C_{\text{pub}}(x)$, and a private part, $C_{\text{priv}}(C_{\text{pub}}(x), y)$.

In their work, Ishai and Wee demonstrated information-theoretic constructions for branching programs and formulas. This line of work was further explored in the context of privacy-free garbled circuits [66, 47, 71], which improve both computational efficiency and size compared to fully hiding garbling. However, these constructions still exhibit linear dependency on the circuit/program size, thus not meeting the strong succinctness goal targeted in this work.

Fully succinct partial garbling schemes for *truth tables* based on one-way functions are implied by [60, 59, 6]. However, these techniques do not generalize to more expressive program classes, nor to *fully hiding* garbling for truth tables.

In contrast, fully hiding succinct garbling schemes have relied on heavy cryptographic tools such as iO or combinations of FHE and ABE [18]. Prior to the current work, succinct partial garbling for general circuits, without resorting to such tools, remained open. Our work addresses this gap by presenting a succinct partial garbling scheme for general circuits from group-based assumptions.

Homomorphic MACs. Gennaro and Wichs [48] constructed fully homomorphic MACs supporting general circuits using FHE. Subsequent works further give practical constructions based

on generic assumptions like the existence of one-way functions, or group assumptions, e.g., [33, 10, 34]. However, the latter constructions support only restricted functions, such as low-degree polynomials. Gorbunov, Vaikuntanathan, and Wichs [56] constructed the stronger *homomorphic signature* primitive from LWE, for bounded-depth circuits. Subsequent constructions from group-based assumptions, such as subexponential DDH or DLin over prime order groups, or k-Lin assumption over bilinear groups were given in [4, 2], where the signature size grows with the depth of computation. There is also a folklore construction that avoids the size dependency on the depth using existing SNARK for P schemes and digital signatures, which can in turn be based on (for example) subexponential DDH, or LWE, or SXDH in asymmetric bilinear groups [38, 39, 37, 93]. The folklore construction is restricted to a one-hop evaluation.

Our algebraic HMAC focuses on an orthogonal dimension, requiring the MAC to have the specific algebraic structure of $\Delta \cdot x + K$. As a result, it can be naturally combined with other cryptographic primitives, such as garbling, HSS, and potentially others. We strongly rely on this feature for obtaining our applications of aHMAC.

Concurrent Work. In an independent and concurrent work [75], Liu, Wang, Yang, and Yu 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.

2 Technical Overview

2.1 Algebraic Homomorphic MACs

In an algebraic homomorphic MAC (aHMAC) scheme, 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:

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

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 [23, 43] present an algorithm, DDLog that works for any cyclic group (of order p) and small exponents $m <$ poly, but fails with 1/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/poly correctness error. (b) The work of [1] 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:

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

Assuming the term $\Delta \cdot z$ is small, we can apply the DDLog guarantee (although with 1/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 4 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 $\mathbf{z} = C(\mathbf{x})$. [8] We implement those features in our final constructions in Section 4, 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.

Comparing aHMAC with HSS. At first glance, our aHMAC scheme may appear similar to a homomorphic secret sharing (HSS) scheme: both schemes consider a pair of parties that initially hold a form input shares of $\mathbf{x}$, and then locally evaluate over those to obtain additive shares of $\mathbf{y} = f(\mathbf{x})$. However, a key difference is that in HSS, *both* parties' computation depend on the inputs, while in aHMAC, only the evaluator's computation depends on the input, but not the authenticator's. As we show below, this difference is what enables the application of aHMAC to garbling schemes, where only the evaluator's computation may depend on the input, but not the garbler's, as well as to constrained PRF schemes, where only the "constrained" evaluation may depend on the input, but not the "unconstrained" evaluation.

In more detail, in HSS the dependency of both parties' computation on the inputs stems from how they evaluate multiplications over shares of two values x and y. When the two parties

[8] 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.

hold additive shares of x and y, it is not clear how to "multiply" them to obtain shares of xy. A key idea in HSS is that if x is an input, and both parties hold (circular) encryptions of $(x \cdot \mathsf{sk})$, these encryptions can be "multiplied" with additive shares of $(y \cdot \mathsf{sk})$ to yield an additive share of $(xy \cdot \mathsf{sk})$.

To avoid the dependency of both parties' computation on the inputs, it is therefore neccessary to come up with a new evaluation method for multiplication. A key technical contribution of this work is a method for "multiplying" additive shares of x, y (without encryptions of x), when x and y are known to just one party (i.e., the evaluator) while the other party (i.e., the authenticator)'s computation is independent of x, y.

2.2 Application: Succinct Partial Garbling for Circuits

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.

$\underline{\mathsf{CDSGb}}(f, y)$:
$\Delta \leftarrow \$,\ (\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \Delta)$
$\sigma_b^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, b, \mathsf{id}^{(i)})$,
$k_f \leftarrow \mathsf{EvalKey}(\mathsf{sk}, f, \{\mathsf{id}^{(i)}\})$,
$\mathsf{ct}_y \leftarrow \mathsf{E.Enc}(\Delta, y)$,
Output $\{L_{x,b}^{(i)} = \sigma_b^{(i)}\}$,
and $\mathsf{gb} = (\mathsf{evk}, k_f, \mathsf{ct}_y)$.

$\underline{\mathsf{CDSEv}}(f, \mathsf{gb}, \mathbf{x}, \{L_x^{(i)} = \sigma_{\mathbf{x}[i]}^{(i)}\})$:
Output 0 if $f(\mathbf{x}) = 0$. O/w:
$\sigma_z \leftarrow \mathsf{EvalTag}(\mathsf{evk}, f, \mathbf{x}, \{\sigma_{\mathbf{x}[i]}^{(i)}\})$,
$\Delta = \sigma_z - k_f$,
$y = \mathsf{E.Dec}(\Delta, \mathsf{ct}_y)$,
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 5.2 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 $\mathbf{z}$.

2.3 Application: Constrained PRF for Circuits

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 [40]. The observation of [40] 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 6 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.

Blueprint:
$\mathsf{msk} = (\mathsf{evk}_0, I_0, \ldots,), \mathsf{sk}_C = (\mathsf{evk}_1, I_1, \ldots,),$
$\mathsf{Eval}:$ derive Δ_0, $z_0 \leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{evk}_0, I_0, \Delta_0, \mathsf{F}_{\mathbf{x}}),$
$\mathsf{CEval}:$ derive Δ_1 $z_1 \leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{evk}_1, I_1, \Delta_1, \mathsf{F}_{\mathbf{x}}).$

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 [40] uses another special HSS to let Eval, CEval derive the desired shares Δ_0, Δ_1. We instead use an aHMAC scheme for this.

CPRF for Circuits from aHMACs and HSS. In our construction, 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 $\Delta_1 = \Delta \cdot C(\mathbf{x}) + \Delta_0$.

3 Preliminaries

3.1 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 [63], 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: $P(\mathbf{x}, \mathbf{y}) = P_{\mathrm{priv}}(P_{\mathrm{pub}}(\mathbf{x}), \mathbf{y})$, 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 $P_{\mathrm{pub}}, P_{\mathrm{priv}}$ 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 8, 10);
- Bounded-depth Boolean circuits: $\mathcal{C}^d = \{\mathcal{C}^d_\lambda\}$, where $\mathcal{C}^d_\lambda$ consists of all two-input Boolean circuits with depth bounded by some fixed polynomial $d(\lambda)$, based on P-DDH and DDH (Definition 7, 6).

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 $2^{\text{poly}(\lambda)}$.
- Quadratic polynomials (mod 2): $\mathcal{Q} = \{\mathcal{Q}_\lambda\}$, where $\mathcal{Q}_\lambda$ consists of quadratic polynomials with below $2^{\text{poly}(\lambda)}$ 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 P, with Boolean inputs, of the form $P(\mathbf{x}, \mathbf{y}) = P_{\text{priv}}(P_{\text{pub}}(\mathbf{x}), \mathbf{y})$, and $\mathcal{L} = \{\mathcal{L}_\lambda\}_\lambda$ be a label space of sizes $|\mathcal{L}_\lambda| \le 2^{\text{poly}(\lambda)}$. 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, 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* $\text{negl}(\lambda)$ *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:*

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

(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 for every* $\lambda \in N$ *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 for all* $\lambda \in N$ *and* $P \in \mathcal{P}_\lambda$*, the garbling size* $\widehat{P}$ *is bounded by* $\mathrm{poly}(\lambda, \ell_z)$*.*

3.2 Cryptographic Assumptions

In this work, we will use two types of groups: (1) groups satisfying the non-interactive distributed log sharing (NIDLS) framework [1], 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 [23, 43].

Definition 4 (NIDLS Framework [1]). *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:*

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

  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 1. Compared to the description in [1], 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 [1], and class groups.

Definition 5 (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 [23, 43]). *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* $m \leq B$*:*

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

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

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

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

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

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 7 (Power-DDH Assumption [31, 7]). *We say the power-DDH assumption holds in the NIDLS framework if the following holds:*

$$\left\{ \mathsf{pp}, (\rho, g), g^s, g^{s^2} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s \leftarrow [\ell]. \end{array} \right\}$$
$$\approx_c \left\{ \mathsf{pp}, (\rho, g), g^a, g^b \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ a, b \leftarrow [\ell]. \end{array} \right\}.$$

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

$$\left\{ G, g, g^s, g^{s^2} \;\middle|\; \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ s \leftarrow \mathbb{Z}_p. \end{array} \right\} \approx_c \left\{ G, g, g^a, g^b \;\middle|\; \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ a, b \leftarrow \mathbb{Z}_p. \end{array} \right\}.$$

Remark 2. In prime-order groups, power-DDH implies DDH: the reduction given a power-DDH tuple (g, g^s, g^{s^2}) samples $a, b \leftarrow \mathbb{Z}_p$ to re-randomize the tuple as $(g, g^{s\cdot a}, g^{s\cdot b}, g^{s^2\cdot ab})$, which becomes a valid DDH tuple. If the reduction is given a random tuple (g, g^s, g^r), the re-randomized is also random. However, in the NIDLS framework there is no clear way to perform this re-randomization.

Definition 8 (Circular-Power-DDH Assumption). *We say the circular-power-DDH assumption holds in the NIDLS framework if the following holds:*

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

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

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

Remark 3. 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 [91].

The following small-exponent assumption is commonly assumed in the NIDLS framework, and is requires for obtaining HSS (for NC1 circuits) in prior work [1]. We don't rely on this assumption except when using existing HSS schemes as a black box.

Definition 9 (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 [83, 41] 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 [83, 41]). 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 10 (KDM Security [17, 20]). *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 4. 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.

Lemma 4 (DDLog algorithm for Damgård-Jurik [90]). *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:*

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

3.3 CP-DDH in Generic Group Model

We first recall the definition of (Shoup's) generic group model [91] as formulated in [96], and then prove that our new assumption, CP-DDH in prime order groups, holds in this model (Theorem 4).

Definition 11 (GGM [91, 96]). *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:

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

□
□

4 Algebraic Homomorphic MACs

Our notion of algebraic homomorphic MACs (aHMACs) can be roughly viewed as a refinement of existing homomorphic MACs (HMACs [3, 48, 33]). In both notions, there are Auth, and EvalTag algorithms that respectively produce authentication tags σ_x for some input x, and homomorphically evaluate some circuit C over the 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.

In Section 4.5, we show that our requirement on the algebraic form of the evaluated tags, together with our simulation-based security definition (Definition 12) implies verifiability and (a weaker variant of) unforgeability as commonly required for HMAC schemes. However, as our applications in this work doesn't require explicitely verifying the evaluated tags, we omit the Verify algorithm, verifiability, and unforgeability from our Definition 12.

Definition 12 (Algebraic Homomorphic MACs). *An algebraic homomorphic MAC scheme 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:*

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

Security: *There exists an efficient simulator* Sim *such that for every sequence of global MAC keys* $\{\boldsymbol{\Delta}_\lambda\}_\lambda$ *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 (suppressing the subscript* λ *for brevity):*

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

Succinctness: *The bit-lengths of tags produced by* Auth *and* $\mathsf{EvalTag}$ *are bounded by some fixed* $\mathrm{poly}(\lambda)$*.*

A leveled aHMAC scheme has the following differences compared to a fully homomorphic scheme (Definition 12):

- KeyGen additionally takes a parameter 1^D where D is a bound on the depth of evaluation circuits.
- $\mathsf{EvalTag}$ 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.)

We omit spelling out the formal definition for the leveled case.

In Section 4.1 and 4.2, we show constructions of aHMACs in the NIDLS framework and in prime-order groups, based on CP-DDH. In Section 4.3, we show leveled variants of these constructions based on P-DDH and DDH. In Section 4.4, we separately describe a (non-leveled) construction based on the KDM security of Damgård-Jurik encryption. While Damgård-Jurik

instantiates NIDLS, the assumption of KDM security is technically different from CP-DDH. In Section 4.5 we show that our aHMAC definition implies verifiability and (a weaker variant of) unforgeability as commonly required for HMAC schemes.

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 4, 8), there exists an aHMAC scheme achieving* negl*-correctness.*
2. *Assuming CP-DDH in prime-order groups, there exists an aHMAC scheme achieving* $1/p$*-correctness for any polynomial* p*.*
3. *Assuming the KDM security (Definition 10) of Damgård-Jurik encryption (Construction 1), there exists an aHMAC scheme achieving* negl*-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 P-DDH and DDH in the NIDLS framework (Definition 7, 6), there exists a leveled aHMAC scheme achieving* negl*-correctness.*
2. *Assuming P-DDH in prime-order groups, there exists a leveled aHMAC scheme achieving* $1/p$*-correctness for any polynomial* p*.*

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

4.1 aHMACs from the NIDLS Framework

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 vectorized operations, 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 $\mathsf{F}^n(s, x) = (\ldots, \mathsf{F}(s, x\|i), \ldots)_{i \in [n]}$.

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

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 $\boldsymbol{\Delta}$ (as a bit matrix in $\{0,1\}^{\ell_z \times \lambda}$):

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

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 $\overline{\mathbf{s}}$ and the (secret) PRF key key_2 from sk. Then compute an authentication tag

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

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$:

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 s_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:

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

   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

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

   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 can verify that 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

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

   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: To help digest the construction, we have broken up and embedded correctness analysis as notes in the above.

Efficiency: We note that the tags output by Auth and the $\mathsf{EvalTag}$ all have bounded magnitude by $O(t) \leq O(2^\lambda)$. 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 a secure aHMAC scheme.*

Proof (of Lemma 5). The security of an aHMAC scheme (Definition 12) requires a simulator Sim to simulate authentication tags $\sigma_{\mathbf{x}} = (\ldots, \sigma_x^{(i)}, \ldots)$ and an evaluation key $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$. It simply samples all of the components at random:

- Sample random authentication tags $\widetilde{\sigma}_x \leftarrow [2^\lambda]^{\ell_s}$.
- Sample a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}_1$.
- Sample public parameters of a NIDLS group $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, and a random group element $(g, \rho) \leftarrow \mathsf{Samp}(\mathsf{pp})$. Then sample random ciphertexts $\mathsf{ct}_s = (\widetilde{\mathbf{h}}, \widetilde{\mathbf{h}}_1, \widetilde{\mathbf{h}}_2)$, and $\mathsf{ct}_\Delta = (\widetilde{\mathbf{H}}, \widetilde{\mathbf{H}}_1)$:

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

We show a series of hybrids that transitions from the real-world distribution in Definition 12 (Hyb_0) to the above simulated distribution (Hyb_5).

Hyb_0 : We summarize the real-world distribution of $\sigma_{\mathbf{x}} = (\ldots, \sigma_x^{(i)}, \ldots)$, and $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}_s = (\mathbf{h}, \mathbf{h}_1, \mathbf{h}_2)$, and $\mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}_1)$.

$$\begin{aligned} &\mathsf{key}_1 \leftarrow \mathcal{K}_1, \quad \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda), && \\ &\mathbf{h} = g^{\mathbf{r}},\ \mathbf{h}_1 = g^{s\cdot\mathbf{r}},\ \mathbf{h}_2 = g^{s^2\cdot\mathbf{r}} \cdot f^{\bar{\mathbf{s}}}, && \left|\begin{array}{l}(\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), \mathbf{r} \leftarrow [\ell]^{\ell_s}, \\ \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, s \leftarrow [\ell].\end{array}\right. \quad (4) \\ &\mathbf{H} = g^{\mathbf{R}},\ \mathbf{H}_1 = g^{s\cdot\mathbf{R}} \cdot f^{\boldsymbol{\Delta}}, && \\ &\sigma_x^{(i)} = \bar{\mathbf{s}} \cdot \mathbf{x}[i] + \mathsf{F}_2^{\ell_s}(\mathsf{key}_2, \mathbf{id}[i]) \text{ over } \mathbb{Z} && |\mathsf{key}_2 \leftarrow \mathcal{K}_2. \quad (5) \end{aligned}$$

Hyb_1 : Instead of computing each tag $\sigma_x^{(i)}$ as in Equation 5, Hyb_1 simulates it as $\widetilde{\sigma}_x^{(i)} \leftarrow [2^\lambda]^{\ell_s}$. 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 4, Hyb_2 simulate it as $\widetilde{\mathbf{R}} = \mathbf{r}' \cdot \mathbf{r}^T$, where $\mathbf{r}' \leftarrow [\ell]^{\ell_z}$. [9] 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:

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

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

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

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

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

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

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 $\boldsymbol{\Delta}$. The Samp algorithm (Definition 4) ensures

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

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.

[9] 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 $\mathbf{r}' \cdot \mathbf{r}^T$.

4.2 aHMACs from 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 $\phi(\cdot) := \mathsf{F}_3(\mathsf{key}_3, \cdot)$. [10]

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 $\bar{\mathbf{s}} \in \{0,1\}^{\ell_s}$) and $\boldsymbol{\Delta}$ (as a bit matrix in $\{0,1\}^{\ell_z \times \lambda}$):

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

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 $\mathsf{sk} = (\mathsf{pp}, \mathbf{h}, \mathbf{H}, \bar{\mathbf{s}}, \mathsf{key}_1, \mathsf{key}_2, \mathsf{key}_3)$, and $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1, \mathsf{key}_3)$.

The proof of the following lemma is completely analogous to that of Lemma 5. We omit writing it out again.

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

4.3 Leveled HAEs without Circular Security Assumptions

In this section, we sketch modifications to Construction 2 and 3, based on the NIDLS framework and prime-order groups respectively, to avoid the circular security assumption, CP-DDH 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 modified construction in the NIDLS framework assumes P-DDH and DDH, and the modified construction in prime-order groups assumes P-DDH.

We focus on explaining the modification to Construction 2. (The modification to Construction 3 is analogous.) Recall that we rely on CP-DDH assumption to argue security of the ciphertext ct_s in the evaluation key evk:

$$\mathsf{ct}_s = (\mathbf{h} = g^{\mathbf{r}}, \quad \mathbf{h}_1 = g^{s \cdot \mathbf{r}}, \quad \mathbf{h}_2 = g^{s^2 \cdot \mathbf{r}} \cdot f^{\bar{\mathbf{s}}}), \text{ where } \mathbf{r}, s \leftarrow \$.$$

In particular, we need the circular assumption because the exponents $\bar{\mathbf{s}}$ in the easy group $\langle f \rangle$ are bit representations of the secret exponent s. The modification is to use a independent secret $\bar{\mathbf{s}}'$ instead:

$$\mathsf{ct}'_s = (\mathbf{h} = g^{\mathbf{r}}, \quad \mathbf{h}_1 = g^{s \cdot \mathbf{r}}, \quad \mathbf{h}'_2 = g^{s^2 \cdot \mathbf{r}} \cdot f^{\bar{\mathbf{s}}'}), \text{ where } \mathbf{r}, s, s' \leftarrow \$.$$

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

P-DDH suffifces for proving the security of ct'_s, but our evaluation procedure (for Mult) now turns two tags σ_x, σ_y under the secret s into a new tag σ_z under the secret s':

$$\text{If } \sigma_x = \bar{\mathbf{s}} \cdot x + \mathbf{k}_x, \quad \sigma_y = \bar{\mathbf{s}} \cdot y + \mathbf{k}_y, \quad \text{then } \sigma_z = \bar{\mathbf{s}}' \cdot z + \mathbf{k}_z.$$

In order to continue evaluting tags under the secret s', we need to provide another ciphertext ct''_s in evk, using the same secret s' as ct'_s and another fresh secret s''.

$$\mathsf{ct}''_s = (\mathbf{h} = g^{\mathbf{r}}, \quad \mathbf{h}_1 = g^{s' \cdot \mathbf{r}}, \quad \mathbf{h}'_2 = g^{s'^2 \cdot \mathbf{r}} \cdot f^{\bar{\mathbf{s}}''}), \text{ where } \mathbf{r}, s'' \leftarrow \$.$$

Generalizing this idea, we can put D such ciphertexts in evk to support evaluations of circuits C with $\mathrm{Depth}(C) \leq D$.

We omit writing out details for the leveled constructions in the NIDLS framework and prime-order groups, as they are mostly analogous to Construction 2 and 3.

4.4 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 use this DDLog variant on ciphertexts encrypting the 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 $\mathbb{Z}$: [11]

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

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 $\mathbb{Z}$.

Security of this construction relies on the KDM security of Damgård-Jurik encryption, which ensures ct_s does not leak anything about sk. We omit writing out details for this construction, as it's mostly analogous to Construction 2.

[11] This usage of DDLog is inspired by the technique from [78].

4.5 Verifiability and Unforgeability of aHMAC

In this section, we describe a simple Verify algorithm using our aHMAC definition (Definition 12) as a black-box, and show that our definition implies verifiability and unforgeability properties.

Compared to the usual unforgeability notion in homomorphic MAC literatures [3, 48, 33], ours is weaker in two aspects: (1) we restrict the adversary to submit authentication and verification queries in one-shot, before seeing the evaluation key evk; (2) we only consider so called "type II" forgeries, where an adversary queries some authenticated data $\mathbf{x}$, and forges authentications of false computation results $\mathbf{z} \neq C(\mathbf{x})$. (The usual notion considers forgeries also of computations results from un-authenticated inputs.)

While our techniques are likely to yield HMAC schemes satisfying the usual, stronger, unforgeability notion, as well as other desired properties like composability, we leave exploring them to future works.

The Verify algorithm. We define the syntax of a Verify algorithm compatible with our aHMAC (Definition 12).

- $\mathsf{Verify}(\boldsymbol{\Delta}, \mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$: takes as inputs the global secret $\boldsymbol{\Delta}$, the secret key sk, a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$, the ids 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.

We implement Verify as follows. As a degenerate case, if C is a constant function, i.e. $C(\cdot) = \mathbf{z}^*$, then directly check whether $\mathbf{z}^* \stackrel{?}{=} \mathbf{z}$. For non-constant C, first compute the evaluated MAC key $\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$. Then check whether the evaluated tags satisfy the correct form: $\sigma_{\mathbf{z}} \stackrel{?}{=} \boldsymbol{\Delta} \odot \mathbf{z} + \mathbf{k}_C$ (over $\mathbb{Z}$). If yes, output $\top$. Otherwise, output $\bot$.

Verifiability. We adapt the formulation from [33] (named authentication and evaluation correctness there) as follows.

Definition 13 (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 $|C| \leq p(\lambda)$, 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*

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

It's clear (hence we omit the proof) that any aHMAC scheme with δ correctness error, and extended with the Verify algorithm described above, satisfies δ-verifiability.

Proposition 1. *An aHMAC scheme with δ-correctness per Definition 12, satisfies δ-verifiability per Definition 13.*

Unforgeability. We adapt the formulation of type-II unforgeability from [33], and restrict the adversary to submit all its authentication and verification queries in one shot, as opposed to adaptively, before seeing the evaluation key evk.

Definition 14 (Non-adaptive Type-II 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*

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

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

1. *Launch $\mathcal{A}(1^\lambda)$. Receive from $\mathcal{A}$ an output length 1^{ℓ_z} and the following:*
   - *authentication queries $\{x^{(i)}, \mathsf{id}^{(i)}\}$ with distinct ids, and*
   - *verification queries $\{C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma^{(i)}_{\mathbf{z}}\}$.*
2. *Sample a global secret $\boldsymbol{\Delta} \leftarrow [2^\lambda]^{\ell_z}$, and run $(\mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta})$. Send to $\mathcal{A}$ the evaluation key evk, as well as answers to the queries 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 *only if both of the following holds.*
   - *It verifies, i.e., $\mathsf{Verify}(\boldsymbol{\Delta}, \mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}}) = \top$.*
   - *All $id^{(i)} \in \mathbf{id}$ are queried in Step 2. Let $\mathbf{x}$ be the queried inputs corresponding to $\mathbf{id}$. We have $C(\mathbf{x}) \neq \mathbf{z}$.*

We show that any aHMAC scheme with negl correctness error also satisfy non-adaptive type-II unforgeability.

Proposition 2. *An aHMAC scheme with* $\mathrm{negl}(\lambda)$*-correctness per Definition 12 satisfy non-adaptive type-II unforgeability per Definition 14.*

Proof. We show a series of hybrids that transitions from $\mathsf{Hyb}_0 := \mathrm{Exp}^{\mathcal{A}}_{\mathrm{HMAC}}(\lambda)$ to Hyb_3, where the the queries of $\mathcal{A}$ are all answered without depending on the global secret $\boldsymbol{\Delta}$. We then argue that a forgery is impossible in Hyb_2 due to the randomness of $\boldsymbol{\Delta}$.

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

Hyb_1: Instead of answering the verification queries using $\mathsf{sk}, \boldsymbol{\Delta}$ and running Verify, the experiment Hyb_1 directly checks constant functions $C^{(i)}$ against $\mathbf{z}^{(i)}$, and answers $\perp$ for non-constant $C^{(i)}$. We claim (and prove in the end) that these answers are correct except with negligible probability, which implies $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Claim. For every efficient adversary $\mathcal{A}$, given no inputs, its probability of forging an evaluated tag w.r.t. a non-constant circuit is negligible:

$$\Pr\left[\begin{array}{l|c} \mathsf{Verify}(\boldsymbol{\Delta}, \mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}}) & \boldsymbol{\Delta} \leftarrow [2^\lambda]^{\ell_z} \\ = \top \text{ AND } C \text{ non-const.} & (\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta}) \\ & (\mathbf{z}, \mathbf{id}, C, \sigma_{\mathbf{z}}) \leftarrow \mathcal{A}(1^\lambda) \end{array}\right] < \mathrm{negl}(\lambda).$$

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

- If exists an $id \in \mathbf{id}$ not queried in Step 2, directly output "Loss".
- Otherwise, let $\mathbf{x}$ be the queried inputs corresponding to $\mathbf{id}$, and $\sigma_{\mathbf{x}}$ be the answers to those queries. If $C(\mathbf{x}) = \mathbf{z}$, directly output "Loss".

– Otherwise, 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".

By negl(λ)-correctness, $\sigma^*_{\mathbf{z}}$ satisfy $\sigma^*_{\mathbf{z}} = \boldsymbol{\Delta} \odot C(\mathbf{x}) + \mathbf{k}_C$. By construction, the forgery passes verification only if $\sigma_{\mathbf{z}} = \boldsymbol{\Delta} \odot \mathbf{z} + \mathbf{k}_C = \sigma^*_{\mathbf{z}} + (\mathbf{z} - C(\mathbf{x})) \odot \boldsymbol{\Delta}$. Hence we have $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_1$.

Note that in Hyb_2, the global secret $\boldsymbol{\Delta}$ and the secret key sk are only used for answering authentication queries, and not for verification.

Hyb_3: Instead of answering authentication queries using $\boldsymbol{\Delta}, \mathsf{sk}$ and running Auth, the experiment Hyb_3 runs $\mathsf{Sim}(1^\lambda, 1^{\ell_z})$ as guaranteed by Definition 12 to simulate the answers, as well as the evaluation key evk. The aHMAC security guarantees $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

Note that in Hyb_3, the global secret $\boldsymbol{\Delta}$ and the secret key sk are not used for answering queries or computing evk.

By a hybrid argument, we conclude that $\mathrm{Exp}^{\mathcal{A}}_{\mathrm{HMAC}}(\lambda) \equiv \mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$. In Hyb_3, the adversary $\mathcal{A}$ wins only if it outputs a forgery $(C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$ such that $\sigma_{\mathbf{z}} = \sigma^*_{\mathbf{z}} + (C(\mathbf{x}) - \mathbf{z}) \odot \boldsymbol{\Delta}$, and $C(\mathbf{x}) - \mathbf{z} \neq \mathbf{0}$. As noted, the adversary $\mathcal{A}$'s view is entirely independent of $\boldsymbol{\Delta}$, hence the forgery has negligible chance of passing the checks due to the randomness of $\boldsymbol{\Delta}$. It remains to prove the claim.

Proof (of Claim). Suppose there exists an adversary $\mathcal{A}$ that creates a successful forgery with non-negligible probability, we construct a reduction $\mathcal{B}$ for aHMAC security (Definition 12) as follows:

– Launch $\mathcal{A}(1^\lambda)$ and obtain a forgery $(\mathbf{z}, \mathbf{id}, C, \sigma_{\mathbf{z}})$.
– Let $\ell_z = \mathsf{Len}(\mathbf{z})$, and sample a global secret $\boldsymbol{\Delta} \leftarrow [2^\lambda]^{\ell_z}$. Choose an arbitrary input vector $\mathbf{x}$, such that $C(\mathbf{x}) \neq \mathbf{z}$.
– Receive from the challenger, w.r.t. $\boldsymbol{\Delta}, \mathbf{x}, \mathbf{id}$, an evaluation key evk and authentication tags $\sigma_{\mathbf{x}}$. Compute $\sigma^*_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$, and check

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

If yes, output 1. Otherwise, output 0.

Note that when the challenger sends correctly computed authentication tags w.r.t. $\boldsymbol{\Delta}, \mathbf{x}, \mathbf{id}$, the check passes if $\mathcal{A}$ has created a successful forgery. On the otherhand, when the challenger sends simulated tags independent of $\boldsymbol{\Delta}$, $\mathbf{x}$, or $\mathbf{id}$, the check passes with negligible probability due to the randomness of $\boldsymbol{\Delta}$. □

□

5 Succinct Partial Garbling

In this section, we show how to construct succinct partial garbling schemes for circuits from aHMACs. In Section 5.1, we show the simpler case using an aHMAC with negl correctness errors. In Section 5.2, 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 7 (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 security of Damgård-Jurik encryption.*

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 8 (Succinct Partial Garbling for Bounded-Depth Circuits). *Let $\mathcal{C}$ be the class of two-input Boolean circuits as in Theorem 7. 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. *P-DDH and DDH in the NIDLS framework;*
2. *P-DDH in prime-order groups.*

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.

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

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.*

5.1 Construction from negl-Correct aHMACs

Construction 4 (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 $C(\mathbf{x},\mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}),\mathbf{y})$. We refer to $C_{\text{pub}} : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_w}$ as the public sub-circuit, and $C_{\text{priv}} : \{0,1\}^{\ell_w} \times \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_z}$, 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 $\widehat{C_{\text{priv}}}$ of the private sub-circuit:

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

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

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

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 $\mathbf{k}_{\text{pub}}, \overline{\mathbf{k}}_{\text{pub}}$ which will be used as "decryption helpers":

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

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

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

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

Parse $\widehat{C_{\text{priv}}}$, $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$, and $\mathsf{evk}, \overline{\mathsf{evk}}, \mathbf{k}_{\text{pub}}, \overline{\mathbf{k}}_{\text{pub}}$ 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}:

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

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

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

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

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 8).

Lemma 7. *Construction 4 is a secure partial garbling scheme.*

5.2 Construction from 1/poly-Correct aHMACs

Definition 15 (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 5. For $t < n/3$, the standard Shamir's secret sharing is also a robust secret sharing. This suffices for our application. For $n/3 \leq t < n/2$, there also exists robust secret sharing schemes with a negligible correctness error and larger share size, e.g. [87, 35].

Construction 5 (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 4, 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 4 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 4 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 4 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 4, 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 8. *Construction 5 is a secure partial garbling scheme.*

Proof (of Lemma 8). 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

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

– 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 $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$.

$$\mathsf{ct}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\Delta_j, L_w^{(j)}) \\ \mathsf{E.Enc}(\Delta_j, 0) \end{cases} \qquad \overline{\mathsf{ct}}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\overline{\Delta}_j, 0) & \text{if } \mathbf{w}[j] = 1 \\ \mathsf{E.Enc}(\overline{\Delta}_j, L_w^{(j)}) & \text{o/w .} \end{cases} \tag{6}$$

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

$$\{\Delta_{j,t}\}_t \leftarrow \begin{cases} \mathsf{Share}(\Delta_j), \\ \mathsf{Share}(0), \end{cases} \qquad \{\overline{\Delta}_{j,t}\}_t \leftarrow \begin{cases} \mathsf{Share}(0) & \text{if } \mathbf{w}[j] = 1, \\ \mathsf{Share}(\overline{\Delta}_j) & \text{o/w.} \end{cases} \tag{7}$$

– Proceed as in Construction 5 to compute $\mathsf{Tables}_t = (\mathsf{evk}_t, \overline{\mathsf{evk}}_t, \mathbf{k}_{\text{pub},t}, \overline{\mathbf{k}}_{\text{pub},t})$ 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

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

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

$$\begin{aligned} &\mathsf{Comps}_{\mathbf{x},C_{\text{pub}}}(\mathsf{Keys} = \{\Delta_j, \overline{\Delta}_j\}_j) : \\ &\text{Denote } \boldsymbol{\Delta} = (\ldots, \Delta_j, \ldots)_j,\ \overline{\boldsymbol{\Delta}} = (\ldots, \overline{\Delta}_j, \ldots)_j, \\ &\text{Compute} \\ &(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta}), \qquad (\overline{\mathsf{sk}}, \overline{\mathsf{evk}}), \leftarrow \mathsf{KeyGen}(1^\lambda, \overline{\boldsymbol{\Delta}}), \\ &\quad \mathbf{k}_{\text{pub}} \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C_{\text{pub}}, \mathbf{id}), \qquad \overline{\mathbf{k}}_{\text{pub}} \leftarrow \mathsf{EvalKey}(\overline{\mathsf{sk}}, \overline{C_{\text{pub}}}, \overline{\mathbf{id}}) \\ &\quad \sigma_x^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \mathbf{id}[i]), \qquad \overline{\sigma}_x^{(i)} \leftarrow \mathsf{Auth}(\overline{\mathsf{sk}}, \mathbf{x}[i], \overline{\mathbf{id}}[i]), \\ &\text{Output } \mathsf{Tables} = (\mathsf{evk}, \overline{\mathsf{evk}}, \mathbf{k}_{\text{pub}}, \overline{\mathbf{k}}_{\text{pub}}), \text{ and } \{L_x^{(i)} = (\sigma_x^{(i)}, \overline{\sigma}_x^{(i)})\}_i. \end{aligned} \tag{8}$$

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 $|f_\lambda| \le p(\lambda)$, 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 8.
Hyb'_1 : Instead of computing $\mathbf{k}_f$, $\overline{\mathbf{k}}_f$ as in Equation 8, Hyb'_1 simulates them as

$$\begin{aligned}
&\sigma_{\mathbf{x}} := (\ldots, \sigma_x^{(i)}, \ldots)_i, && \overline{\sigma}_{\mathbf{x}} := (\ldots, \overline{\sigma}_x^{(i)}, \ldots)_i,\\
&\sigma_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{evk}, f, \mathbf{x}, \sigma_{\mathbf{x}}), && \overline{\sigma}_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\overline{\mathsf{evk}}, \overline{f}, \mathbf{x}, \overline{\sigma}_{\mathbf{x}}),\\
&\mathbf{k}_f[j] \leftarrow \begin{cases} \sigma_{\mathbf{w}}[j] - \Delta_j, \\ \sigma_{\mathbf{w}}[j], \end{cases} && \overline{\mathbf{k}}_f[j] \leftarrow \begin{cases} \overline{\sigma}_{\mathbf{w}}[j], & \text{if } \mathbf{w}[j] = 1, \\ \overline{\sigma}_{\mathbf{w}}[j] - \overline{\Delta}_j, & \text{o/w}, \end{cases}
\end{aligned} \tag{9}$$

where $\mathbf{w} = f(\mathbf{x})$. The $1/(2\lambda)$-correctness of HAE (Definition 12) 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 8, 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 HAE 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 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 9. 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

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

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

We will then use the following lemma from [25], which is further based on a computational hardcore lemma from [77]. 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 9 (Simulating Leaky Functions [25]). *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* $m \in \mathcal{M}$,

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

– $\mathsf{SimLeaky}$ *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 $\widehat{C} = (\widehat{C_{\text{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \{\mathsf{Tables}_t\})$ and labels $\{L_x^{(i)} = (\ldots, L_{x,t}^{(i)}, \ldots)\}$, $\{L_y^{(i)}\}$.

$$\begin{array}{l|l} L_y^{(i)} = K_y^{(i)}(\mathbf{y}[i]), \quad \widehat{C_{\text{priv}}}, & \\ \mathsf{ct}_j \leftarrow \mathsf{E.Enc}(\Delta_j, K_w^{(j)}(1)), & \widehat{C_{\text{priv}}}, \{K_w^{(j)}\}, \{K_y^{(i)}\} \leftarrow \mathsf{BG.Garb}(h, \{K_z^i\}), \\ \overline{\mathsf{ct}}_j \leftarrow \mathsf{E.Enc}(\overline{\Delta}_j, K_w^{(j)}(0)), & \qquad \Delta_j, \overline{\Delta}_j \leftarrow \{0,1\}^\lambda \end{array} \tag{10}$$

$$\begin{array}{l|l} (\{L_{x,t}^{(i)}\}, \mathsf{Tables}_t) & \{\Delta_{j,t}\} \leftarrow \mathsf{Share}(\Delta_j),\ \{\overline{\Delta}_{j,t}\} \leftarrow \mathsf{Share}(\overline{\Delta}_j) \\ \quad \leftarrow \mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}_t). & \mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_j \end{array} \tag{11}$$

Hyb_1 : Instead of computing $\{L_{x,t}^{(i)}\}, \mathsf{Tables}_t$ as in Equation 8, Hyb_1 simulates them using the $\mathsf{Erase}_{1/\lambda}$ and $\mathsf{SimLeaky}$ algorithms from Lemma 9:

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

Lemma 9 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Proceeds as in Hyb_1, but aborts if there are $\geq \lambda$ 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 8, Hyb_3 simulates them according to $\mathbf{w}$ as in Equation 7.

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 15) ensures $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 : Change back to computing $(\{L_{x,t}^{(i)}\}, \mathsf{Tables}_t)$ from $\mathsf{Comps}_{\mathbf{x},C_{\text{pub}}}$ as in Equation 8, instead of using Erase_λ and $\mathsf{SimLeaky}$, as they are potentially inefficient. (The shares are still simulated as in Equation 7.)

Lemma 9 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 10, simulate them as in Equation 6 (re-created here), where $L_w^{(j)} := K_w^{(j)}[\mathbf{w}[j]]$.

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

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

Hyb_6 : Instead of computing $\widehat{C_{\text{priv}}}$, $\{L_y^{(i)}\}$ and $\{L_w^j = K_w^{(j)}[\mathbf{w}[j]]\}$ as in Equation 10, simulate them using the simulator $\mathsf{BG.Sim}$ guaranteed by the security of Boolean garbling:

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

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. □

5.3 Implications: Succinct Secret Sharing, Garbling, and PSM

We first point out an immediate implication to succinct secret sharing for partite functions (See [6] 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. [12]

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 7:*

1. *CP-DDH in either the NIDLS framework or prime-order groups;*
2. *the KDM security of Damgård-Jurik encryption.*

There also exists a scheme for partite functions specified as bounded-depth (and unbounded size) Boolean circuits, where the share sizes are $\text{poly}(\lambda)$ *bits, assuming any for the assumptions from Theorem 8:*

1. *P-DDH and DDH in the NIDLS framework;*
2. *P-DDH in prime-order groups.*

[12] 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.

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 [54]. 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 16 (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 10 (HE Schemes). *There exist the following constructions:*

1. *[62, 44] 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. *[19] 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 9 (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 10 (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 [46], 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,* [13] *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 9 and 10 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 8;*
- *The DCR assumption, or DDH in prime-order groups;*
- *The subgroup decision problem in bilinear groups of composite order.*

6 Application: 1-Key Selective CPRF for Circuits

The notion of constrained PRFs (CPRF) is first proposed (concurrently) in [21, 67, 26], where besides the usual pair of algorithms KeyGen and Eval for PRFs, there exists another pair Constrain and CEval. 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 [30, 32, 28, 9, 36, 84, 40] 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.

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

Definition 17 (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 18. We recap the observation of [40] 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 $\mathbf{\Delta} \cdot w$ is a valid memory share of w, where $\mathbf{\Delta}$ is a secret vector in the HSS scheme.

Definition 18 (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* $\mathbf{\Delta} \in [2^\lambda]^{\ell_d}$*.*
- $\mathsf{Input}(\mathsf{pk}, x \in \mathcal{I})$ *takes as inputs the public key* pk *and an input* x*. It outputs a pair of input shares* I_0, I_1*.*
- $\mathsf{ExtEval}(\beta \in \{0,1\}, \mathsf{evk}_\beta, \mathbf{I}_\beta, \mathbf{\Delta}_\beta, P \in \mathcal{P})$ *takes as inputs a party identity* β*, its evaluation key* evk_β *and input shares* $\mathbf{I}_\beta$*, its share of the extension secret* $\mathbf{\Delta}_\beta$*, 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 $|P| \le p(\lambda)$, any inputs $\mathbf{x} \in \mathcal{I}^n$, any extension bit $w \in \{0,1\}$, and any share vector $\mathbf{\Delta}_0 \in \mathbb{Z}^{\ell_d}$, the following holds:*

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

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 $\beta \in \{0,1\}$:*

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

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 $(\mathsf{pk}, \mathsf{evk}_0, \mathsf{evk}_1, \mathbf{\Delta}) \leftarrow \mathsf{Setup}(1^\lambda)$ 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 6. 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 $\mathbf{\Delta}$ as its overall evaluation key: $\mathsf{evk}_\beta^* := (\mathsf{evk}_\beta, \mathbf{\Delta}_\beta)$. 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 11 (Extended HSS). *There exists an extended HSS scheme for NC1 assuming either of the following:*

1. *[81] The DCR assumption.*
2. *[1] DDH and the small exponent assumption (Definition 9) in the NIDLS framework.*

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

Theorem 11 (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. *P-DDH, DDH and the small exponent assumption in the NIDLS framework.*
2. *the KDM security of Damgård-Jurik encryption.*

Construction 6 (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:

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

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

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

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

$$\mathbf{\Delta}_0 := \mathbf{k}_U \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{sk}, U_{\mathbf{x}}, \mathbf{id}),^{14}$$

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

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

[14] 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.

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

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

Deterministically derive distinct $\mathbf{id}$ 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 $\mathbf{\Delta}_1$ of the extension secret:

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

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

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

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

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 12. *Construction 6 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 17.

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

- 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:

$$\begin{aligned} &(\boxed{\mathsf{HSS.pk}}, \mathsf{HSS.evk}_0, \boxed{\mathsf{HSS.evk}_1}, \mathbf{\Delta}) \leftarrow \mathsf{HSS.Setup}(1^\lambda), \\ &\mathbf{s} \leftarrow \mathcal{I}^{\ell_k}, \quad (I_0^{(i)}, \boxed{I_1^{(i)}}) \leftarrow \mathsf{HSS.Input}(\mathsf{HSS.pk}, \mathbf{s}[i]), \end{aligned} \tag{12}$$

$$\begin{aligned} &(\mathsf{aHMAC.sk}, \boxed{\mathsf{aHMAC.evk}}) \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \mathbf{\Delta}), \\ &\boxed{\sigma^{(i)}} \leftarrow \mathsf{aHMAC.Auth}(\mathsf{aHMAC.sk}, C[i], \mathbf{id}[i]), \end{aligned} \tag{13}$$

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

$$\begin{aligned} \mathbf{I}_0 &= (\ldots, I_0^{(i)}, \ldots), \\ \mathbf{\Delta}_0 = \mathbf{k}_U &\leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{aHMAC.sk}, U_{\mathbf{x}}, \mathbf{id}), \\ \boxed{z_0} &\leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{HSS.evk}_0, \mathbf{I}_0, \mathbf{\Delta}_0, \mathsf{F}_{\mathbf{x}}). \end{aligned} \tag{14}$$

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 14, 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 13, 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 12, 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.

References

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 421–452. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15985-5˙15
2. Afshar, A., Cheng, J., Goyal, R.: Leveled fully-homomorphic signatures from batch arguments. Cryptology ePrint Archive, Report 2024/931 (2024), https://eprint.iacr.org/2024/931
3. Agrawal, S., Boneh, D.: Homomorphic MACs: MAC-based integrity for network coding. In: Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D. (eds.) ACNS 09International Conference on Applied Cryptography and Network Security. LNCS, vol. 5536, pp. 292–305. Springer, Berlin, Heidelberg (Jun 2009). https://doi.org/10.1007/978-3-642-01957-9˙18
4. Anthoine, G., Balbás, D., Fiore, D.: Fully-succinct multi-key homomorphic signatures from standard assumptions. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part III. LNCS, vol. 14922, pp. 317–351. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68382-4˙10
5. Applebaum, B., Arkis, B., Raykov, P., Vasudevan, P.N.: Conditional disclosure of secrets: Amplification, closure, amortization, lower-bounds, and separations. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 727–757. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7˙24
6. Applebaum, B., Beimel, A., Ishai, Y., Kushilevitz, E., Liu, T., Vaikuntanathan, V.: Succinct computational secret sharing. In: Saha, B., Servedio, R.A. (eds.) 55th ACM STOC. pp. 1553–1566. ACM Press (Jun 2023). https://doi.org/10.1145/3564246.3585127
7. Applebaum, B., Harnik, D., Ishai, Y.: Semantic security under related-key attacks and applications. In: Chazelle, B. (ed.) ICS 2011. pp. 45–60. Tsinghua University Press (Jan 2011)
8. Assouline, L., Liu, T.: Multi-party PSM, revisited. Cryptology ePrint Archive, Report 2019/657 (2019), https://eprint.iacr.org/2019/657
9. Attrapadung, N., Matsuda, T., Nishimaki, R., Yamada, S., Yamakawa, T.: Constrained PRFs for NC^1 in traditional groups. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 543–574. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0˙19
10. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 863–874. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516681
11. Bartusek, J., Ma, F., Zhandry, M.: The distinction between fixed and random generators in group-based assumptions. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 801–830. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26951-7˙27
12. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC. pp. 503–513. ACM Press (May 1990). https://doi.org/10.1145/100216.100287
13. Beimel, A., Farràs, O., Nir, O.: Secret-sharing schemes for high slices. Cryptology ePrint Archive, Paper 2024/602 (2024), https://eprint.iacr.org/2024/602
14. Beimel, A., Ishai, Y., Kumaresan, R., Kushilevitz, E.: On the cryptographic complexity of the worst functions. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 317–342. Springer, Berlin, Heidelberg (Feb 2014). https://doi.org/10.1007/978-3-642-54242-8˙14
15. Beimel, A., Kushilevitz, E., Nissim, P.: The complexity of multiparty PSM protocols and related models. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 287–318. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8˙10
16. Bitansky, N., Canetti, R., Garg, S., Holmgren, J., Jain, A., Lin, H., Pass, R., Telang, S., Vaikuntanathan, V.: Indistinguishability obfuscation for RAM programs and succinct randomized encodings. SIAM J. Comput. **47**(3), 1123–1210 (2018). https://doi.org/10.1137/15M1050963, https://doi.org/10.1137/15M1050963
17. Black, J., Rogaway, P., Shrimpton, T.: Encryption-scheme security in the presence of key-dependent messages. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 62–75. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/3-540-36492-7˙6
18. Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8441, pp. 533–556. Springer (2014). https://doi.org/10.1007/978-3-642-55220-5“˙30, https://doi.org/10.1007/978-3-642-55220-5_30
19. Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Berlin, Heidelberg (Feb 2005). https://doi.org/10.1007/978-3-540-30576-7˙18

20. Boneh, D., Halevi, S., Hamburg, M., Ostrovsky, R.: Circular-secure encryption from decision Diffie-Hellman. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 108–125. Springer, Berlin, Heidelberg (Aug 2008). https://doi.org/10.1007/978-3-540-85174-5˙7
21. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer, Berlin, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42045-0˙15
22. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: Optimizations and applications. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2105–2122. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134107
23. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4˙19
24. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: Optimizing rounds, communication, and computation. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 163–193. Springer, Cham (Apr / May 2017). https://doi.org/10.1007/978-3-319-56614-6˙6
25. Boyle, E., Gilboa, N., Ishai, Y., Kolobov, V.I.: Programmable distributed point functions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 121–151. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15985-5˙5
26. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer, Berlin, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0˙29
27. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 3–33. Springer, Cham (May 2019). https://doi.org/10.1007/978-3-030-17656-3˙1
28. Brakerski, Z., Tsabary, R., Vaikuntanathan, V., Wee, H.: Private constrained PRFs (and more) from LWE. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 264–302. Springer, Cham (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2˙10
29. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS. pp. 97–106. IEEE Computer Society Press (Oct 2011). https://doi.org/10.1109/FOCS.2011.12
30. Brakerski, Z., Vaikuntanathan, V.: Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 1–30. Springer, Berlin, Heidelberg (Mar 2015). https://doi.org/10.1007/978-3-662-46497-7˙1
31. Camenisch, J., Neven, G., shelat, a.: Simulatable adaptive oblivious transfer. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 573–590. Springer, Berlin, Heidelberg (May 2007). https://doi.org/10.1007/978-3-540-72540-4˙33
32. Canetti, R., Chen, Y.: Constraint-hiding constrained PRFs for NC^1 from LWE. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 446–476. Springer, Cham (Apr / May 2017). https://doi.org/10.1007/978-3-319-56620-7˙16
33. Catalano, D., Fiore, D.: Practical homomorphic MACs for arithmetic circuits. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 336–352. Springer, Berlin, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9˙21
34. Catalano, D., Fiore, D., Gennaro, R., Nizzardo, L.: Generalizing homomorphic MACs for arithmetic circuits. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 538–555. Springer, Berlin, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0˙31
35. Cevallos, A., Fehr, S., Ostrovsky, R., Rabani, Y.: Unconditionally-secure robust secret sharing with compact shares. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 195–208. Springer, Berlin, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4˙13
36. Chen, Y., Vaikuntanathan, V., Wee, H.: GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 577–607. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0˙20
37. Choudhuri, A.R., Garg, S., Jain, A., Jin, Z., Zhang, J.: Correlation intractability and SNARGs from subexponential DDH. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 635–668. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3˙20
38. Choudhuri, A.R., Jain, A., Jin, Z.: Non-interactive batch arguments for NP from standard assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 394–423. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8˙14

39. Choudhuri, A.R., Jain, A., Jin, Z.: SNARGs for $\mathcal{P}$ from LWE. In: 62nd FOCS. pp. 68–79. IEEE Computer Society Press (Feb 2022). https://doi.org/10.1109/FOCS52979.2021.00016
40. Couteau, G., Meyer, P., Passelègue, A., Riahinia, M.: Constrained pseudorandom functions from homomorphic secret sharing. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part III. LNCS, vol. 14006, pp. 194–224. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30620-4˙7
41. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Berlin, Heidelberg (Feb 2001). https://doi.org/10.1007/3-540-44586-2˙9
42. Davidson, A., Katsumata, S., Nishimaki, R., Yamada, S., Yamakawa, T.: Adaptively secure constrained pseudorandom functions in the standard model. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 559–589. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2˙19
43. Dinur, I., Keller, N., Klein, O.: An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 213–242. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0˙8
44. Döttling, N., Garg, S., Ishai, Y., Malavolta, G., Mour, T., Ostrovsky, R.: Trapdoor hash functions and their applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 3–32. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8˙1
45. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith III, W.E.: Homomorphic secret sharing from paillier encryption. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) ProvSec 2017. LNCS, vol. 10592, pp. 381–399. Springer, Cham (Oct 2017). https://doi.org/10.1007/978-3-319-68637-0˙23
46. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: 26th ACM STOC. pp. 554–563. ACM Press (May 1994). https://doi.org/10.1145/195058.195408
47. Frederiksen, T.K., Nielsen, J.B., Orlandi, C.: Privacy-free garbled circuits with applications to efficient zero-knowledge. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 191–219. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6˙7
48. Gennaro, R., Wichs, D.: Fully homomorphic message authenticators. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 301–320. Springer, Berlin, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42045-0˙16
49. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press (May / Jun 2009). https://doi.org/10.1145/1536414.1536440
50. Gentry, C., Ramzan, Z.: Single-database private information retrieval with constant communication rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer, Berlin, Heidelberg (Jul 2005). https://doi.org/10.1007/11523468˙65
51. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4˙5
52. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. J. Comput. Syst. Sci. **60**(3), 592–629 (2000). https://doi.org/10.1006/JCSS.1999.1689, https://doi.org/10.1006/jcss.1999.1689
53. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th FOCS. pp. 464–479. IEEE Computer Society Press (Oct 1984). https://doi.org/10.1109/SFCS.1984.715949
54. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 555–564. ACM Press (Jun 2013). https://doi.org/10.1145/2488608.2488678
55. Golle, P., Jarecki, S., Mironov, I.: Cryptographic primitives enforcing communication and storage complexity. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 120–135. Springer, Berlin, Heidelberg (Mar 2003). https://doi.org/10.1007/3-540-36504-4˙9
56. Gorbunov, S., Vaikuntanathan, V., Wichs, D.: Leveled fully homomorphic signatures from standard lattices. In: Servedio, R.A., Rubinfeld, R. (eds.) 47th ACM STOC. pp. 469–477. ACM Press (Jun 2015). https://doi.org/10.1145/2746539.2746576
57. Goyal, R., Hohenberger, S., Koppula, V., Waters, B.: A generic approach to constructing and proving verifiable random functions. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part II. LNCS, vol. 10678, pp. 537–566. Springer, Cham (Nov 2017). https://doi.org/10.1007/978-3-319-70503-3˙18
58. Gueron, S., Lindell, Y., Nof, A., Pinkas, B.: Fast garbling of circuits under standard assumptions. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 567–578. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813619

59. Heath, D.: Efficient arithmetic in garbled circuits. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part V. LNCS, vol. 14655, pp. 3–31. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58740-5˙1
60. Heath, D., Kolesnikov, V.: One hot garbling. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 574–593. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484764
61. Hsieh, Y.C., Lin, H., Luo, J.: Attribute-based encryption for circuits of unbounded depth from lattices. In: 64th FOCS. pp. 415–434. IEEE Computer Society Press (Oct 2023). https://doi.org/10.1109/FOCS57990.2023.00031
62. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer, Berlin, Heidelberg (Feb 2007). https://doi.org/10.1007/978-3-540-70936-7˙31
63. Ishai, Y., Wee, H.: Partial garbling schemes and their applications. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part I. LNCS, vol. 8572, pp. 650–662. Springer, Berlin, Heidelberg (Jul 2014). https://doi.org/10.1007/978-3-662-43948-7˙54
64. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. In: Khuller, S., Williams, V.V. (eds.) 53rd ACM STOC. pp. 60–73. ACM Press (Jun 2021). https://doi.org/10.1145/3406325.3451093
65. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part I. LNCS, vol. 13275, pp. 670–699. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-06944-4˙23
66. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 955–966. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516662
67. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 669–684. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516668
68. Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: Flexible garbling for XOR gates that beats free-XOR. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 440–457. Springer, Berlin, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44381-1˙25
69. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Berlin, Heidelberg (Jul 2008). https://doi.org/10.1007/978-3-540-70583-3˙40
70. Komargodski, I., Yogev, E.: Another step towards realizing random oracles: Non-malleable point obfuscation. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 259–279. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9˙10
71. Kondi, Y., Patra, A.: Privacy-free garbled circuits for formulas: Size zero and information-theoretic. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 188–222. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7˙7
72. Koppula, V., Lewko, A.B., Waters, B.: Indistinguishability obfuscation for Turing machines with unbounded memory. In: Servedio, R.A., Rubinfeld, R. (eds.) 47th ACM STOC. pp. 419–428. ACM Press (Jun 2015). https://doi.org/10.1145/2746539.2746614
73. Kushilevitz, E., Ostrovsky, R.: Replication is NOT needed: SINGLE database, computationally-private information retrieval. In: 38th FOCS. pp. 364–373. IEEE Computer Society Press (Oct 1997). https://doi.org/10.1109/SFCS.1997.646125
74. Lipmaa, H.: An oblivious transfer protocol with log-squared communication. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3650, pp. 314–328. Springer (2005). https://doi.org/10.1007/11556992“˙23, https://doi.org/10.1007/11556992_23
75. Liu, H., Wang, X., Yang, K., Yu, Y.: Garbled circuits with 1 bit per gate. Cryptology ePrint Archive, Paper 2024/1988 (2024), https://eprint.iacr.org/2024/1988
76. Liu, T., Vaikuntanathan, V., Wee, H.: Conditional disclosure of secrets via non-linear reconstruction. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 758–790. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7˙25
77. Maurer, U.M., Tessaro, S.: A hardcore lemma for computational indistinguishability: Security amplification for arbitrarily weak PRGs with optimal stretch. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 237–254. Springer, Berlin, Heidelberg (Feb 2010). https://doi.org/10.1007/978-3-642-11799-2˙15
78. Meyer, P., Orlandi, C., Roy, L., Scholl, P.: Rate-1 arithmetic garbling from homomorphic secret-sharing. IACR Cryptol. ePrint Arch. p. 820 (2024), https://eprint.iacr.org/2024/820

79. Meyer, P., Orlandi, C., Roy, L., Scholl, P.: Rate-1 arithmetic garbling from homomorphic secret sharing. In: Boyle, E., Mahmoody, M. (eds.) Theory of Cryptography. pp. 71–97. Springer Nature Switzerland, Cham (2025)
80. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Feldman, S.I., Wellman, M.P. (eds.) Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999. pp. 129–139. ACM (1999). https://doi.org/10.1145/336992.337028, https://doi.org/10.1145/336992.337028
81. Orlandi, C., Scholl, P., Yakoubov, S.: The rise of paillier: Homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 678–708. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5˙24
82. Ostrovsky, R., Skeith III, W.E.: A survey of single-database private information retrieval: Techniques and applications (invited talk). In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 393–411. Springer, Berlin, Heidelberg (Apr 2007). https://doi.org/10.1007/978-3-540-71677-8˙26
83. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer, Berlin, Heidelberg (May 1999). https://doi.org/10.1007/3-540-48910-X˙16
84. Peikert, C., Shiehian, S.: Privately constraining and programming PRFs, the LWE way. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 675–701. Springer, Cham (Mar 2018). https://doi.org/10.1007/978-3-319-76581-5˙23
85. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Berlin, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7˙15
86. Quach, W., Waters, B., Wichs, D.: Targeted lossy functions and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 424–453. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8˙15
87. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: Johnson, D.S. (ed.) Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA. pp. 73–85. ACM (1989). https://doi.org/10.1145/73007.73014, https://doi.org/10.1145/73007.73014
88. Ragavan, S., Vafa, N., Vaikuntanathan, V.: Indistinguishability obfuscation from bilinear maps and lpn variants. In: Boyle, E., Mahmoody, M. (eds.) Theory of Cryptography. pp. 3–36. Springer Nature Switzerland, Cham (2025)
89. Rosulek, M., Roy, L.: Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 94–124. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0˙5
90. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 687–717. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84252-9˙23
91. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Berlin, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0˙18
92. Stern, J.P.: A new efficient all-or-nothing disclosure of secrets protocol. In: Ohta, K., Pei, D. (eds.) ASIACRYPT'98. LNCS, vol. 1514, pp. 357–371. Springer, Berlin, Heidelberg (Oct 1998). https://doi.org/10.1007/3-540-49649-1˙28
93. Waters, B., Wu, D.J.: Batch arguments for NP and more from standard bilinear group assumptions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 433–463. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4˙15
94. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986). https://doi.org/10.1109/SFCS.1986.25
95. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6˙8
96. Zhandry, M.: To label, or not to label (in generic groups). In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part III. LNCS, vol. 13509, pp. 66–96. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15982-4˙3