

Beyond DCR: HSS and PCFs from Subgroup Indistinguishability

Sebastian Hasler 

Aarhus University
`sebastian@hasler.dev`

Abstract. We construct homomorphic secret sharing (HSS) and pseudorandom correlation functions (PCFs) in a general group-theoretic framework, with security based on the subgroup indistinguishability (SgI) assumption introduced by Brakerski and Goldwasser (Crypto 2010). Under certain instantiations of this framework, SgI corresponds to decisional composite residuosity (DCR), but other instantiations are possible as well. Hence, our work expands the set of assumptions that imply HSS and PCFs.

Our constructions crucially rely on the existence of an efficient distributed discrete logarithm (DDLog) algorithm for the utilized group. We construct a new DDLog algorithm that works in general groups as long as the order t of the base is smooth. In particular, our DDLog supports the prime-power case, which was previously an open problem. Moreover, we apply a divide-and-conquer optimization that significantly improves the performance from $\mathcal{O}(\log^2 t / \log \log t)$ group operations (Abram et al., Crypto 2022) to $\mathcal{O}(\log t \log \log t)$.

Toward constructing PCFs, we introduce a new notion of public-coin subgroup indistinguishability (PC-SgI). We show that, in certain instantiations, SgI implies PC-SgI, so no new assumption is required. We then obtain PCFs for VOLE, OT, OLE, and degree-two correlations, all generically under PC-SgI. Our PCFs for OLE and degree-two correlations additionally rely on the sparse LPN assumption.

Along the way, we discover and patch two security issues in prior work: first, an attack on decisional Diffie–Hellman (DDH) in the Joye–Libert instantiation by Abram et al. (Crypto 2022), and second, a flaw in the security proofs of optimized variants of the IKNP OT extension protocol and of many silent OT extension protocols.

Keywords: Homomorphic Secret Sharing · Correlated Pseudorandomness · Subgroup Indistinguishability · Distributed Discrete Logarithm

This report is the full version of our ASIACRYPT 2026 publication [Has26].

Table of Contents

1	Introduction	3
2	Technical Overview	5
2.1	HSS from Subgroup Indistinguishability	6
2.2	New DDLog Algorithm	8
2.3	PCFs from Subgroup Indistinguishability	9
2.4	Extension to Oblivious Transfer	10
2.5	PCFs for OLE and Degree-Two Correlations	10
3	Notation	10
4	The Subgroup Framework	10
4.1	Hardness Assumptions	11
4.2	Instantiations	14
5	Public-Coin Sampling for Joye–Libert	16
5.1	Number-Theoretic Preliminaries	16
5.2	Our Sampling Algorithm	17
6	Attacks Based on the Power Residue Symbol	19
6.1	Countermeasures	20
7	New DDLog Algorithm	20
8	Generic HSS Template	22
8.1	Subtractive Secret Sharing	22
8.2	Distributed Decryption	22
8.3	RMS Programs	23
8.4	The Template	23
9	HSS from Subgroup Indistinguishability	25
9.1	Basic Cryptosystem	25
9.2	ElGamal-style Cryptosystem	26
9.3	BHHO-style Cryptosystem	28
9.4	Comparison of HSS Instantiations	30
10	PCF Definitions	31
10.1	Weak and Strong PCFs	31
11	PCFs from Subgroup Indistinguishability	34
11.1	PCF for (Module) VOLE	34
11.2	PCF for Random 1-out-of- t Oblivious Transfer	37
11.3	Public-Key PCFs	42
11.4	PCF for OLE and Degree-Two Correlations	43
11.5	Key Sizes and Runtimes	45
	References	48
A	Power Residue Symbols for Further Prime Powers	51

1 Introduction

Homomorphic secret sharing (HSS) allows two parties to compute a public function on secret-shared data without interaction. As such, it is a relaxation of fully homomorphic encryption (FHE): The computation can be performed locally, but is distributed across two non-colluding parties. Therefore, HSS may be easier to obtain (i.e., from weaker assumptions) or more efficient than FHE. Applications of HSS include secure two-party computation and outsourced computation.

In [BGI16a], Boyle et al. construct the first HSS protocol that does not rely on FHE. Their protocol relies only on decisional Diffie–Hellman (DDH), an assumption not known to imply FHE, and supports the class of polynomial-size restricted multiplication straight-line (RMS) programs, a function class that includes the class of polynomial-size branching programs, which in turn includes NC^1 [Bar86].

The main tool from [BGI16a] enabling HSS is a distributed discrete logarithm (DDLog) algorithm that converts multiplicative shares of f^x , where f is some designated group element, into subtractive shares of x . Unfortunately, this DDLog algorithm has an error probability that cannot be made negligible and requires a polynomial-size plaintext space. As a consequence, their HSS protocol needs to repeat the homomorphic RMS program evaluation multiple times in order to correct for the error, leading to a large overhead. Even after optimizations and extensions from [BGI17, FGJS17, BCG⁺17, DKK18], the overhead due to the correctness error could not be eliminated. In [OSY21, RS21], these limitations were overcome by using the Paillier or Damgård–Jurik cryptosystem based on the decisional composite residuosity (DCR) assumption. The crucial insight was that these DCR-based cryptosystems admit a DDLog procedure with perfect correctness and exponential-size plaintext space, which leads to HSS with negligible error probability. In [ADOS22], the restriction to Paillier and Damgård–Jurik (if one aims for HSS with negligible error probability) was lifted by presenting a general group-theoretic framework, which we henceforth call the *subgroup framework*, that can be instantiated to obtain HSS with negligible error probability under the DDH assumption and a small exponent assumption.

While HSS enables secure two-party computation with sublinear communication, all above-mentioned constructions are still computationally expensive and therefore lead to a large latency. As an alternative, one can use HSS in the preprocessing phase, where the latency is less of a concern, to generate (large amounts of) correlated randomness. This is captured by the pseudorandom correlation function (PCF) primitive introduced in [BCG⁺20a]: A PCF allows two parties holding short correlated keys to locally and repeatedly sample correlated randomness. There is a generic approach for constructing a PCF from any HSS scheme that is sufficiently powerful to evaluate a weak pseudorandom function (WPRF) [BCG⁺20a, HRK25]. However, the PCFs obtained via this approach are rather inefficient, since they require evaluating the circuit of a WPRF under HSS for every query.

In [OSY21], Orlandi, Scholl, and Yakoubov observe that the same techniques that yield HSS can also be used to construct PCFs more directly, making only black-box use of a WPRF. Specifically, they obtain *public-key PCFs*¹ for vector oblivious linear evaluation (VOLE) and oblivious transfer (OT), as well as (non-public-key) PCFs for oblivious linear evaluation (OLE) and degree-two correlations (such as Beaver triples). However, the resulting PCFs in [OSY21] only produce secret shares over the rings $\mathbb{Z}_N$ (where N is an RSA semiprime) or $\mathbb{Z}_2$. Moreover, their PCF for OLE and degree-two correlations relies additionally on the variable-density LPN (VDLPN) assumption – the non-standard learning parity with noise (LPN) variant from [BCG⁺20a].

¹ In a public-key PCF, parties can locally generate a public–secret key pair and, after the public keys have been exchanged, any two parties can locally combine their own secret key with the other party’s public key to derive correlated PCF keys.

Subgroup Indistinguishability. In [BG10], Brakerski and Goldwasser introduce a group-theoretic framework, which is similar to the subgroup framework mentioned above, and define the subgroup indistinguishability (SgI) assumption in this framework. Under suitable instantiations of this framework, SgI coincides with quadratic residuosity (QR) or DCR, but other instantiations are possible as well, e.g., the Joye–Libert instantiation and the class group instantiation described in [ADOS22].

In this work, we explore how HSS and PCFs can be based generically on the SgI assumption, in order to expand the set of assumptions that imply these primitives. That is, we generalize the HSS and PCF constructions from [OSY21] to arbitrary instantiations of the subgroup framework, with security based on the SgI assumption.

Along the way, we find an attack against DDH in the Joye–Libert instantiation from [ADOS22], which breaks their cryptosystem. Our definition of the Joye–Libert instantiation contains a countermeasure preventing this class of attacks.

New HSS Instantiations. We present three SgI-based cryptosystems, each of which can be used to instantiate a generic HSS template. The first generically captures cryptosystems such as Paillier [Pai99], Damgård–Jurik [DJ01], and Joye–Libert [JL13], where a ciphertext consists of a single group element. The second is an ElGamal-style cryptosystem similar to the one in [ADOS22], but with security based on SgI instead of DDH. The third is a variant of the circular-secure cryptosystems from [BH00, BG10], which uniquely enables HSS under SgI without an additional assumption, i.e., without any circular security or small exponent assumption. In Table 1.1, we summarize the cryptographic assumptions required by different HSS instantiations in the subgroup framework. As indicated in the table, most of the HSS instantiations support reusing the public group parameters; i.e., the parameters can be generated once in a trusted setup and reused by many pairs of parties. This also enables a single-round HSS setup between any pair of parties, as shown in [ADOS22].

Table 1.1. Comparison of generic HSS instantiations in the subgroup framework

Cryptosystem	Assumptions	Parameter reuse & single-round setup
Basic (Construction 9.2)	SgI + circular security	✗
ElGamal-style [ADOS22]	DDH + small exponent	✓
ElGamal-style (Construction 9.3)	SgI + small exponent	✓
BH00-style (Construction 9.5)	SgI	✓

New DDLog Algorithm. To be compatible with our HSS template, an instantiation of the subgroup framework must possess an efficient DDLog algorithm. Such an algorithm exists in the Paillier and Damgård–Jurik instantiations [OSY21, RS21], and in *any* instantiation where the order t of the base f is a product of small distinct primes [ADOS22]. The prime-power case was left as an open problem in [ADOS22]. We resolve this problem by constructing an efficient DDLog that works for any t that has no superpolynomial-size prime factor. This enables, for the first time, the use of an exponential-size power-of-two plaintext modulus in DDLog applications, such as HSS. Moreover, we apply a divide-and-conquer optimization, which together with the support for smoother t improves the asymptotic performance from $\mathcal{O}(\log^2 t / \log \log t)$ group operations in [ADOS22] to $\mathcal{O}(\log t \log \log t)$. In our HSS evaluation that uses a 384-bit modulus t , the two improvements combined lead to a measured 16× speedup.

New PCFs. We generalize the PCFs from [OSY21] to arbitrary instantiations of the subgroup framework. First, we observe that the SgI assumption is insufficient to prove the security of these PCFs, since they involve ciphertexts sampled from public randomness. To address this issue, we newly introduce the public-coin subgroup indistinguishability (PC-SgI) assumption, which ensures that these ciphertexts hide their underlying plaintext. In certain instantiations of the subgroup framework, the PC-SgI assumption follows from the (standard) SgI assumption. This is the case in the Goldwasser–Micali and Paillier instantiations, which is why [OSY21] did not face the same issue. Moreover, we show that this is also the case in the Joye–Libert instantiation with plaintext modulus $t \mid 6983776800 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$, thereby extending the set of plaintext moduli supported by this type of PCF.

Under the PC-SgI assumption, we immediately obtain a public-key PCF for VOLE via the same construction as in [OSY21]. This can be extended to a public-key PCF for OT using a standard technique involving a correlation-robust hash function (CRHF). However, we observe that the security proofs of many prior works applying this technique do not go through from correlation robustness alone [ALSZ13a, KK13, BCG⁺19b, BCG⁺19a, BCG⁺20a, OSY21, ADOS22, BCG⁺22]. We resolve this issue via a slightly modified protocol and corrected security proof. Moreover, in the extension to OLE and degree-two correlations [OSY21], we replace VDLPN with the well-established sparse LPN assumption.

Our SgI-based PCFs achieve a lower throughput than existing PCFs from LPN variants [BCG⁺22, HR26]; in exchange, however, the SgI-based PCFs feature vastly smaller key sizes: only a few KiB for VOLE, a few dozen KiB for OT, and a few dozen MiB for OLE or degree-two correlations. We report the key sizes and the measured runtimes in Section 11.5.

List of Contributions.

- We introduce the new notion of PC-SgI (Section 4) and show that SgI implies PC-SgI in certain small-modulus Joye–Libert instantiations (Section 5).
- We show a higher-residuosity attack on DDH in the Joye–Libert variant from [ADOS22] which breaks their cryptosystem (Section 6). We also suggest a countermeasure preventing this class of attacks.
- We present an efficient DDLog algorithm for subgroups of arbitrary smooth order, including the important and previously unsolved prime-power case (Section 7).
- We show that HSS can be generically constructed under the SgI assumption using any of our three SgI-based cryptosystems, and we compare the share sizes and measured runtimes of these instantiations (Section 9).
- We construct public-key PCFs for VOLE generically under the PC-SgI assumption (Section 11.1).
- We discover a flaw in the security proofs of the optimized IKNP OT extension protocol from [ALSZ13a, KK13] and of the silent OT extension protocols from [BCG⁺19b, BCG⁺20a], and we give a modified protocol whose security we prove from correlation robustness (Section 11.2).
- We construct PCFs for OLE and degree-two correlations generically under the conjunction of the PC-SgI and sparse LPN assumptions (Section 11.4).

2 Technical Overview

Background on HSS. An RMS program (cf. Definition 8.2) is an arithmetic circuit where multiplication gates are restricted so that one of the two operands must be an input value. Two-party HSS constructions for polynomial-size RMS programs typically represent an input value x as an linearly homomorphic encryption (LHE) ciphertext $I_x := \text{Enc}(\text{pk}, x \cdot \text{sk})$, and a memory value y as subtractive shares $M_{y,\sigma} := \llbracket y \cdot \text{sk} \rrbracket_{\sigma}^{(\mathbb{Z})}$ held by party P_{σ} for $\sigma \in \{0, 1\}$.

Here, the superscript $(\mathbb{Z})$ denotes that these are subtractive shares *over the integers*. An input value can be multiplied by a memory value via a *distributed decryption* algorithm DDec that implements the following functionality: Two parties, holding a ciphertext $\text{ct} = \text{Enc}(\text{pk}, x)$ and subtractive shares of $y \cdot \text{sk}$ can locally compute subtractive shares of the plaintext x , i.e.:

$$\text{DDec}(\llbracket y \cdot \text{sk} \rrbracket_1^{(\mathbb{Z})}, \text{ct}) - \text{DDec}(\llbracket y \cdot \text{sk} \rrbracket_0^{(\mathbb{Z})}, \text{ct}) \equiv \text{Dec}(\text{sk}, \text{ct}) \cdot y \pmod{t}. \quad (2.1)$$

Many group-based LHE schemes encode a plaintext $x \in \mathbb{Z}_t$ in a cyclic group $F = \langle f \rangle$ of order t where the discrete logarithm is easy, i.e., x is encoded as f^x . To decrypt a ciphertext ct , one first uses the secret key to recover f^x and then computes the discrete logarithm with base f to obtain x . Examples of such cryptosystems are Paillier [Pai99] where $\text{Dec}(\text{sk}, c) = \log_f(c^{\text{sk}})$ with $f = 1 + N$, and ElGamal [ElG85] where $\text{Dec}(\text{sk}, (c_0, c_1)) = \log_f(c_0^{-\text{sk}} \cdot c_1)$.

With these cryptosystems, two parties holding a ciphertext $\text{Enc}(\text{pk}, x)$ and secret key shares $\llbracket \text{sk} \rrbracket^{(\mathbb{Z})}$ can use their respective share to locally compute *divisive* shares a_0, a_1 of f^x , i.e., party P_σ (for $\sigma \in \{0, 1\}$) obtains a_σ and it holds that $a_1 = a_0 f^x$. Moreover, since the decryption procedure is linear in sk ,² one can insert a multiplicative factor $y \in \mathbb{Z}$, i.e., when using shares $\llbracket y \cdot \text{sk} \rrbracket^{(\mathbb{Z})}$ instead, the parties obtain divisive shares of $f^{x \cdot y}$.

The final ingredient for a distributed decryption is a DDLog algorithm to locally convert divisive shares of f^x into subtractive shares of $x \pmod{t}$. Formally, a DDLog algorithm satisfies the following identity for inputs $a_0, a_1 \in G$ with $a_1 = a_0 f^x$:

$$\text{DDLog}(\text{pp}, 1, a_1) - \text{DDLog}(\text{pp}, 0, a_0) \equiv x \pmod{t}.^3$$

Here, pp are public parameters containing at least a description of a group G , the base element $f \in G$, and the order $t = |\langle f \rangle|$.

By locally computing $\text{DDec}(M_{y,\sigma}, I_x)$, the parties can obtain subtractive shares of $x \cdot y \cdot \text{sk}$ over $\mathbb{Z}_t$. If the plaintext modulus t is large enough (i.e., much larger than an upper bound on $x \cdot y \cdot \text{sk}$), then, with high probability, these are also correct shares of $x \cdot y \cdot \text{sk}$ over the integers, i.e., a memory value of $x \cdot y$, which can be used in further multiplications by input values.

2.1 HSS from Subgroup Indistinguishability

[ADOS22] introduced a general group-theoretic framework that we refer to as the *subgroup framework*. This framework captures settings where the group G is defined as the inner direct product of two subgroups F and H (i.e., $G = FH \cong F \times H$). As above, F is required to be cyclic, t denotes its order, and f denotes a designated generator of F . This framework can be instantiated with the groups used by many cryptosystems, such as Goldwasser–Micali [GM82], Paillier [Pai99], Damgård–Jurik [DJ01], Joye–Libert [JL13], and more.

[ADOS22] constructs an ElGamal-style cryptosystem based on the DDH assumption within the subgroup framework. Their DDH variant asserts that

$$(\text{pp}, g, g^x, g^y, g^{xy}) \approx_c (\text{pp}, g, g^x, g^y, g^z),$$

where pp are public parameters related to the group G , $g \in G$ is a random group element with $f \in \langle g \rangle$, and x, y, z are random exponents. The ElGamal-style cryptosystem over a suitable group allows for the instantiation of HSS with an exponential-size plaintext space and negligible error probability.

² In the ElGamal case, for the decryption to be linear in sk , we consider a two-component secret key $\text{sk} = (s_0, s_1) = (d, 1)$ and define $\text{Dec}((s_0, s_1), (c_0, c_1)) := c_0^{-s_0} \cdot c_1^{s_1}$.

³ Some works such as [BGI16a] only demand this to hold with high probability. For simplicity, our definition only covers perfectly correct DDLog algorithms, as this is sufficient for our purposes.

In our work, we expand the set of assumptions that imply HSS by replacing DDH with the SgI assumption introduced in [BG10]. The SgI assumption asserts that

$$(\text{pp}, h) \approx_c (\text{pp}, g),$$

where $h \xleftarrow{\$} H$ and $g \xleftarrow{\$} G$. When using the Paillier or Damgård–Jurik instantiation of the subgroup framework, the SgI assumption coincides with the DCR assumption and the resulting HSS schemes mostly coincide with those of [OSY21, RS21]. However, other instantiations are viable as well, including the Joye–Libert instantiation and the class group instantiation described in [ADOS22].

Remark. In the Goldwasser–Micali instantiation, SgI corresponds to QR. However, we do *not* obtain HSS from QR, since the cryptosystems in the Goldwasser–Micali instantiation have plaintext modulus $t = 2$, which is too small. The generic HSS template (cf. Section 8) requires an exponential-size plaintext space.

Cryptosystem Choices. To obtain HSS from subgroup indistinguishability, we require an SgI-based cryptosystem that supports distributed decryption as in Eq. (2.1). We present three cryptosystems generically within the subgroup framework. If the subgroup framework is instantiated with a group $G = FH$ that admits an efficient DDLog, our cryptosystems support distributed decryption. Under the additional condition that $t = |F|$ (which will correspond to the plaintext modulus in our cryptosystems) is exponentially large, any one of the three cryptosystems can be used to instantiate HSS.

1. *Basic Cryptosystem.* This system generically captures cryptosystems like Goldwasser–Micali, Joye–Libert, Paillier, and Damgård–Jurik where a ciphertext consists of a single group element: the encoded message f^x masked by a random element from H . The secret key is only upper-bounded by the order of G and hence does not fit into the plaintext space. Consequently, in the HSS construction based on this cryptosystem, input shares consist of several ciphertexts due to requiring a digit decomposition of the secret key.
2. *ElGamal-style Cryptosystem.* This system has the same features as the generic ElGamal-style cryptosystem from [ADOS22], but with security based on SgI instead of DDH. The secret key sk is a random exponent, the public key is $\text{pk} = (h, h^{-\text{sk}})$, and encryption is defined as $\text{Enc}(\text{pk}, x) = (h^r, (h^{-\text{sk}})^r f^x)$ where r is a random exponent. In the context of HSS, this cryptosystem is usually used with a small exponent assumption, which states that, even when sk is chosen from an interval much smaller than the group order, it still holds that $(h, h^{-\text{sk}}) \approx_c (h, h')$ where $h, h' \xleftarrow{\$} H$. Under this assumption, sk can be chosen small enough to fit entirely into the plaintext space, which has two advantages for HSS: First, input shares no longer need to use a digit decomposition of sk , and second, the circular security assumption is no longer required. The group parameters for this cryptosystem can be generated once in a trusted setup and reused by many parties, enabling single-round distributed key generation.
3. *BHHO-style Cryptosystem.* This is a variant of the cryptosystems from [BHHO08, BG10], which uniquely enables HSS under SgI without any additional assumption, i.e., without any circular security or small exponent assumption. The secret key consists of n small exponents $\text{sk} = (d_1, \dots, d_n)$ and the public key is $\text{pk} = (h_0, h_1, \dots, h_n)$, where $h_1, \dots, h_n \xleftarrow{\$} H$ and $h_0 := \prod_{i=1}^n h_i^{-d_i}$. Encryption is defined as $\text{Enc}(\text{pk}, x) = (h_0^r f^x, h_1^r, \dots, h_n^r)$, where r is a random exponent. With $n = 1$, this is the same as the ElGamal-style cryptosystem. However, n can be chosen sufficiently large such that security follows from the leftover hash lemma, without the small exponent assumption. Note that [BHHO08, BG10] restricted the individual exponents d_i to be from $\{0, 1\}$, while we allow a larger range (still smaller than the plaintext space, so that HSS requires no digit decomposition and no circular

security assumption). This allows us to choose n much smaller than in [BH00, BG10], vastly reducing the ciphertext size. For instance, in a Paillier group with a 3072-bit N , we can use $n = 2$. Consequently, a ciphertext consists of just three group elements – an over 1000-fold reduction compared to [BG10], and only one group element more than an ElGamal-style ciphertext.

2.2 New DDLog Algorithm

Above, we required the existence of an efficient DDLog such that our cryptosystems support distributed decryption. Whether an efficient DDLog exists depends on the concrete instantiation of the subgroup framework. An efficient DDLog exists in the Paillier instantiation [OSY21], as well as in *any* instantiation where the order t of f is the product of small distinct primes [ADOS22]. However, [ADOS22] left the prime-power case as an open problem. We resolve this open problem by constructing an efficient DDLog that works for any t that has no superpolynomial-size prime factor. Our new ingredient is a DDLog algorithm that handles the prime-power $t = p^e$ case (as long as p is polynomial-size). This can then be generalized to products of prime powers using the Chinese remainder theorem (CRT) as in [ADOS22].

Our algorithm requires $\mathcal{O}(\log t \log \log t)$ group operations for sufficiently smooth t , while the one from [ADOS22] requires $\mathcal{O}(\log^2 t / \log \log t)$. This improvement is enabled by the support for much smoother t (e.g., prime powers) and by a divide-and-conquer approach that reduces the number of group operations compared to an iterative approach. Any one of those two optimizations alone would not suffice to improve over $\mathcal{O}(\log^2 t / \log \log t)$.

Our DDLog algorithm closely follows a recursive variant⁴ of the Pohlig–Hellman algorithm from [Sho05] for computing discrete logarithms in F . In the prime-power ($t = p^e$) case, given input $a = f^x$, that algorithm recursively computes the lower and upper halves of the digit decomposition of x via two recursive calls. This works roughly as follows: Let $t_L := p^{\lfloor e/2 \rfloor}$ and $t_U := p^{\lceil e/2 \rceil}$ (hence $t = t_L t_U$). First, compute $a' := a^{t_U}$. Observe that $\omega := f^{t_U}$ generates a subgroup of order t_L and we have $a' = \omega^{x \bmod t_L}$. By recursively calling the algorithm, find $x' = x \bmod t_L$, i.e., the lower half of the digit decomposition of x . Then compute $a f^{-x'}$ and solve its discrete logarithm in the subgroup $\langle f^{t_L} \rangle$ recursively to find the remaining digits, i.e., the upper half.

We observe that this algorithm can be generalized to the DDLog setting, where we have two parties that are given a_0, a_1 with $a_1 = a_0 f^x$: When $e = 1$, the parties use the known DDLog algorithm for polynomial-size subgroups from [ADOS22]. When $e > 1$, the parties start by computing $a'_\sigma := a_\sigma^{t_U}$. Observe that $a'_1 = a'_0 \omega^x$, i.e., they now have divisive shares of ω^x . By recursively calling the DDLog algorithm, the parties obtain x'_0, x'_1 with $x'_1 - x'_0 \equiv x \pmod{t_L}$. At first glance, we are now stuck, because this does not constitute a sharing of $x' \bmod t_L$ over $\mathbb{Z}_t$, since the subtraction might underflow. However, this problem can actually be ignored, i.e., we can simply reinterpret x'_0, x'_1 as shares over $\mathbb{Z}_t$ and proceed analogously to the Pohlig–Hellman algorithm, i.e., the parties next compute $\hat{a}_\sigma := a_\sigma f^{-x'_\sigma}$. Observe that $\hat{a}_1 = \hat{a}_0 f^{x - (x'_1 - x'_0)}$, i.e., they now have divisive shares of $f^{x - (x'_1 - x'_0)}$. Furthermore, $x - (x'_1 - x'_0)$ is divisible by t_L , so the parties can use $\hat{a}_0, \hat{a}_1$ as inputs to the DDLog in the subgroup $\langle f^{t_L} \rangle$ to obtain subtractive shares of $(x - (x'_1 - x'_0))/t_L$. Using the linearity of subtractive shares, they can finally compute shares of x .

In the non-prime-power case, we use a similar divide-and-conquer approach that halves the number of prime powers with each recursion level. This reduces the number of group operations compared to [ADOS22] by reusing exponentiations across different branches of the recursion tree.

⁴ The original algorithm from [PH78] uses an iterative approach that is slower than the recursive variant from [Sho05] explained here.

2.3 PCFs from Subgroup Indistinguishability

We generalize the PCF for VOLE from [OSY21] to arbitrary instantiations of the subgroup framework. The VOLE correlation over $\mathbb{Z}_t$ is defined as follows: Party P_1 holds a fixed secret $\Delta \in \mathbb{Z}$ (called VOLE secret) and the correlation outputs $((a, z_0), z_1)$ where $a, z_0 \xleftarrow{\$} \mathbb{Z}_t$ and $z_1 := z_0 + a\Delta \bmod t$ (hence z_0, z_1 are subtractive shares of $a\Delta \bmod t$).

The PCF for VOLE from [OSY21] uses Paillier encryption, i.e., the group is $G = \mathbb{Z}_{N^2}^*$ and the plaintext space is $\mathbb{Z}_t = \mathbb{Z}_N$, where N is an RSA semiprime. The PCF keys are $k_0 = (\text{sk}, \llbracket \Delta \cdot \text{sk} \rrbracket_0^{(\mathbb{Z})})$ and $k_1 = (\Delta, \llbracket \Delta \cdot \text{sk} \rrbracket_1^{(\mathbb{Z})})$, where sk is a Paillier decryption key. To evaluate the PCF (i.e., to sample from the VOLE correlation), the parties use public randomness to sample a group element g (which is always a valid Paillier ciphertext encrypting some $a := \text{Dec}(\text{sk}, g)$) and then compute $\text{DDec}(\llbracket \Delta \cdot \text{sk} \rrbracket_\sigma^{(\mathbb{Z})}, g)$ to obtain subtractive shares of $\Delta \cdot a \bmod N$. P_0 who knows sk can additionally compute a .

At first sight, this PCF can immediately be generalized to arbitrary instantiations of the subgroup framework by using our basic cryptosystem. However, the security of this PCF crucially depends on the condition that the public randomness used to sample g does not reveal the plaintext a . Whether this is the case depends on the sampling algorithm. For instance, when we sample $\tilde{f} \xleftarrow{\$} F$ and $h \xleftarrow{\$} H$ separately, then $g := \tilde{f}h$ is uniformly random over G (and hence a valid ciphertext), but the randomness directly reveals the corresponding plaintext $a = \log_f(\tilde{f})$. In other words, the SgI assumption is not necessarily sufficient to obtain a PCF for VOLE in arbitrary instantiations of the subgroup framework.

Public-Coin Subgroup Indistinguishability. To resolve this issue, we introduce a public-coin variant of SgI. It is not immediately obvious how to define such a public-coin variant. Recall that the SgI assumption asserts that $(\text{pp}, h) \approx_c (\text{pp}, g)$, where $h \xleftarrow{\$} H$ and $g \xleftarrow{\$} G$. In a public-coin setting, we cannot use two different sampling algorithms for h and g , because it would be trivial to distinguish which algorithm was used by rerunning the algorithms using the public randomness. The essence of SgI is that g hides its projection to F (denoted $\pi_F(g)$), i.e., $(\text{pp}, g, \pi_F(g)) \approx_c (\text{pp}, g, \tilde{f})$ for an independent $\tilde{f} \xleftarrow{\$} F$. This is actually an equivalent way to define the SgI assumption, and this viewpoint allows us to introduce a public-coin variant (called PC-SgI), which underlies our PCFs. That is, the PC-SgI assumption asserts that

$$(\text{pp}, (r_i, g_i, \pi_F(g_i))_{i \in [m]}) \approx_c (\text{pp}, (r_i, g_i, \tilde{f}_i)_{i \in [m]})$$

for any polynomial number m of samples, where $\tilde{f}_i \xleftarrow{\$} F$, and r_i is the randomness used to sample g_i . Our PCFs need this multi-sample form, and, unlike for SgI, it does not follow from the single-sample form.

The validity of the PC-SgI assumption depends on the specific sampling algorithm used to sample g . As noted above, the natural method of sampling from F and H separately and multiplying the outputs is insecure. To base security of our PCFs on SgI instead of PC-SgI, one can employ an *invertible* sampling algorithm where valid randomness can be efficiently simulated given the algorithm's output. We prove that, when such a sampling algorithm is used, SgI implies PC-SgI.

Note that an invertible sampling algorithm does not necessarily exist for all subgroup framework instantiations. For Paillier, an invertible sampling algorithm exists: Simply sample the binary representation of an integer from $[N^2]$ and interpret it as a group element.⁵ This is the sampling algorithm that was implicitly used in [OSY21]. For the Joye–Libert instantiation, we introduce an invertible sampling algorithm based on computing the t^{th} power residue symbol. Consequently, our VOLE PCF can be instantiated securely with any

⁵ A uniformly random element from $[N^2]$ is with overwhelming probability in $\mathbb{Z}_{N^2}^*$. To achieve probability 1, one can repeat the process until the sampled element is coprime to N .

plaintext modulus t that supports an efficient algorithm for computing the t^{th} power residue symbol. This includes configurations where $t \mid 6983776800 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$. In contrast, [OSY21] supported only the cases $t \in \{2, N\}$.

2.4 Extension to Oblivious Transfer

The IKNP OT extension protocol [IKNP03] uses a CRHF to convert VOLE samples to random OTs. The IKNP protocol was optimized in [ALSZ13a, KK13] by using *pseudorandom* VOLE samples instead of truly random ones, thereby reducing communication by roughly a factor of two. Inspired by these protocols, pseudorandom correlation generators (PCGs) or PCFs for random OT are commonly obtained by first evaluating a PCG/PCF for VOLE and then converting the VOLE samples to random OTs via a CRHF [BCG⁺19b, BCG⁺19a, BCG⁺20a, OSY21, ADOS22, BCG⁺22].

We discover that, even in the semi-honest setting, the notion of CRHFs is insufficient to prove the security of all above-mentioned OT protocols [ALSZ13a, KK13, BCG⁺19b, BCG⁺19a, BCG⁺20a, OSY21, ADOS22, BCG⁺22] that are based on pseudorandom VOLE samples. The respective security proofs in [ALSZ13a, ALSZ17, BCG⁺19b] are flawed in the case of a corrupted receiver. Specifically, they involve a hybrid argument where pseudorandom samples are replaced by truly random ones (to make them compatible with the CRHF definition) in a context where the adversary knows the pseudorandom generator (PRG) seed and can therefore distinguish these hybrids.⁶ (The original IKNP protocol from [IKNP03] is not affected.) We are not aware of an attack on any of these protocols; our observation concerns the proofs.

We modify the protocol so that the proof goes through and thus obtain a PCF for random 1-out-of- t OT under the PC-Sgl assumption, where t is the order of F . We refer to Section 11.2 for details.

2.5 PCFs for OLE and Degree-Two Correlations

Finally, we construct a PCF for OLE and additive degree-two correlations over $\mathbb{Z}_t$ by compressing the setup phase of multiple parallel VOLE PCF instances. To this end, we modify the sparse LPN-based PCF for VOLE from [BCM⁺25] so that it generates shares over the integers rather than within a specific modulus. We refer to Section 11.4 for details.

3 Notation

We denote the security parameter by $\lambda \in \mathbb{N}$. When we write $X = X(\lambda)$, this signifies that the object X is a function of the security parameter and that we henceforth leave this dependence implicit. For a finite set X , we denote sampling an element x uniformly at random from X by $x \xleftarrow{\$} X$. We denote identical distributions by $\equiv$, statistical indistinguishability by $\approx_s$, and computational indistinguishability by $\approx_c$. For $t \in \mathbb{N}$, let $[t] := \{0, \dots, t-1\}$, let $\mathbb{Z}_t := \mathbb{Z}/t\mathbb{Z}$, and let $\mathbb{Z}_t^*$ be its group of units.

4 The Subgroup Framework

We restate the group-theoretic framework from [ADOS22], which we henceforth call the *subgroup framework*. A similar framework has been used in [BG10].

Definition 4.1 (Subgroup Framework). *An instantiation of the subgroup framework is a tuple $\mathcal{I}$ that consists of the following probabilistic and polynomial-time (ppt.) algorithms:*

⁶ We have notified the authors about these flaws and some of the corresponding ePrint reports [ALSZ13b, ALSZ16] have been updated.

- A ppt. algorithm $\mathcal{I}.\text{Gen}(1^\lambda)$ that outputs $\text{pp} = (G, H, f, t, \hat{N})$ subject to the following conditions: $G = FH \cong F \times H$ is a finite abelian group that is the inner direct product of two subgroups F, H , where F is cyclic with generator f and order $t \geq 2$. $\hat{N}$ is an upper bound on $|G|$ such that, for any $g \in G$ and random $r \xleftarrow{\$} [\hat{N}]$, g^r is statistically close to uniform over $\langle g \rangle$.
- Optionally, a ppt. algorithm $\mathcal{I}.\text{TrapdoorGen}(1^\lambda)$ that outputs $(\text{pp}, |H|)$ where pp is the same as in the previous algorithm.
- A ppt. algorithm $\mathcal{I}.\text{Sample}(H)$ that outputs h distributed statistically close to uniform over H .
- Optionally, a ppt. algorithm $\mathcal{I}.\text{SampleDDH}(\text{pp})$ that outputs some $g \in G$.
- Optionally, a ppt. algorithm $\mathcal{I}.\text{SampleG}(\text{pp})$ that outputs g distributed statistically close to uniform over G .
- Optionally, a ppt. algorithm $\mathcal{I}.\text{DDLog}(\text{pp}, \sigma \in \{0, 1\}, a \in G)$ as defined in [Definition 4.2](#).

Whenever G or H appear as inputs to or outputs of an algorithm, they are understood to be succinct descriptions of the respective groups. We assume that one can efficiently compute group operations, given such a description.

The algorithm $\mathcal{I}.\text{SampleDDH}$ is used to define the DDH assumption. Note that $g \leftarrow \mathcal{I}.\text{SampleDDH}(\text{pp})$ is not necessarily uniform over G . The algorithms $\mathcal{I}.\text{Sample}$ and $\mathcal{I}.\text{SampleG}$ are additions by us: $\mathcal{I}.\text{Sample}$ is used in our cryptosystems based on the SgI assumption, while $\mathcal{I}.\text{SampleG}$ is used to define the PC-SgI assumption, which underlies our PCFs. An element from G can always be sampled by separately sampling from H (via $\mathcal{I}.\text{Sample}(H)$) and F (by exponentiating f to a random power), but doing so would result in insecure PCFs. Instead, $\mathcal{I}.\text{SampleG}$ should somehow sample from G directly. We discuss this further in [Section 4.1](#), where we formally define the PC-SgI assumption.

The DDLog allows two parties who hold divisive shares a_0, a_1 of an element $f^x \in F$ to convert these into subtractive shares of x . Formally, this is defined as follows:

Definition 4.2 (Distributed Discrete Logarithm). A group generation algorithm Gen that outputs public parameters pp – describing a group G and a base element $f \in G$ of order $t \in \mathbb{Z}_{\geq 2}$ – is said to have an efficient DDLog if there exists an efficient deterministic algorithm $\text{DDLog}(\text{pp}, \sigma \in \{0, 1\}, a \in G)$ that outputs some element from $[t]$ such that, for all pp in the range of Gen , all $x \in [t]$, and all $a_0, a_1 \in G$ with $a_1 = a_0 f^x$, it holds that

$$\text{DDLog}(\text{pp}, 1, a_1) - \text{DDLog}(\text{pp}, 0, a_0) \equiv x \pmod{t}.$$

HSS in the Subgroup Framework. In [\[ADOS22\]](#), Abram et al. construct HSS from any instantiation of the subgroup framework that satisfies the following conditions:

- t (used as the plaintext modulus in the utilized encryption scheme) is sufficiently large.
- The instantiation has an efficient DDLog algorithm.
- The DDH and small exponent assumptions hold (cf. [Definitions 4.3](#) and [4.10](#)).

Our work shows that the DDH assumption can be replaced by SgI. The small exponent assumption is optional in our work and leads to performance improvements.

4.1 Hardness Assumptions

Here, we give definitions of the hardness assumptions that are used in this work and establish two foundational results for our new PC-SgI assumption. Let $\mathcal{I}$ be an instantiation of the subgroup framework in all of the following definitions. Of course, whether or not these assumptions hold depends on the concrete instantiation $\mathcal{I}$.

Decisional Diffie–Hellman. The DDH assumption in [ADOS22] is defined w.r.t. some sampling algorithm (in this work called $\mathcal{I}.\text{SampleDDH}$) for sampling some element $g \in G$. In particular, the constructions in [ADOS22] require that, with overwhelming probability, $f \in \langle g \rangle$. Our definition of the DDH assumption is identical to the one from [ADOS22], except that we do not include the randomness used by $\mathcal{I}.\text{SampleDDH}$ in the distribution, i.e., our DDH assumption is not public-coin. This makes our assumption weaker and our result stronger, since we break the DDH assumption for a certain instantiation of the subgroup framework.

Definition 4.3 (DDH Assumption). *Let $\mathcal{I}$ be a subgroup framework instantiation with an $\mathcal{I}.\text{SampleDDH}$ algorithm. The DDH assumption for $\mathcal{I}$ states that*

$$(\text{pp}, g, g^x, g^y, g^{xy}) \approx_c (\text{pp}, g, g^x, g^y, g^z)$$

where $\text{pp} = (G, H, f, t, \hat{N}) \leftarrow \mathcal{I}.\text{Gen}(1^\lambda)$, $g \leftarrow \mathcal{I}.\text{SampleDDH}(\text{pp})$, and $x, y, z \xleftarrow{\$} [\hat{N}]$.

Subgroup Indistinguishability. The notion of *subgroup indistinguishability* from [BG10] captures a class of decision problems where the adversary’s goal is to distinguish a random group element ($g \xleftarrow{\$} G$) from a random subgroup element ($h \xleftarrow{\$} H$). This captures, e.g., QR [GM82] and DCR [Pai99]. Under such a subgroup indistinguishability assumption, h can be used as a mask to hide an element from F . The framework used in [BG10] is identical to the subgroup framework, except that [BG10] additionally requires the orders of F and H to be coprime. Note that this restriction is not necessary to define subgroup indistinguishability, so we can generalize subgroup indistinguishability to the subgroup framework:

Definition 4.4 (Subgroup Indistinguishability Assumption). *The subgroup indistinguishability (SgI) assumption for $\mathcal{I}$ states that*

$$(\text{pp}, h) \approx_c (\text{pp}, g)$$

where $\text{pp} = (G, H, f, t, \hat{N}) \leftarrow \mathcal{I}.\text{Gen}(1^\lambda)$, $h \xleftarrow{\$} H$, and $g \xleftarrow{\$} G$.

Remark. In typical instantiations (e.g., in all instantiations considered in this work as well as [BG10, ADOS22]), the orders of F and H are coprime. In this case, it holds that $H = \{x^t \mid x \in G\}$, so the SgI assumption coincides with what’s sometimes called the t^{th} *residuosity assumption*.

For our PCFs, where the random coins used to sample an element from G are drawn from public randomness, the SgI assumption is insufficient. To resolve this issue, we need a public-coin variant of SgI. It is not immediately obvious how to define such a public-coin variant. The definition cannot use two different sampling algorithms for h and g , when the randomness is public, because it would be trivial to distinguish which algorithm was used. The essence of SgI is that g hides $\pi_F(g)$, i.e., $(\text{pp}, g, \pi_F(g)) \approx_c (\text{pp}, g, \tilde{f})$ for an independent $\tilde{f} \xleftarrow{\$} F$. This is actually an equivalent way to define the SgI assumption, and this viewpoint allows us to introduce a public-coin variant, which underlies our PCFs:

Definition 4.5 (Public-Coin Subgroup Indistinguishability Assumption). *Let $\mathcal{I}$ be a subgroup framework instantiation with a sampling algorithm $\mathcal{I}.\text{SampleG}$, and let $\ell_r \in \text{poly}(\lambda)$ be such that $\mathcal{I}.\text{SampleG}$ uses at most ℓ_r bits of randomness, written $\mathcal{I}.\text{SampleG}^r$ when the coins are fixed to r . The public-coin subgroup indistinguishability (PC-SgI) assumption for $\mathcal{I}$ states that, for every $m \in \text{poly}(\lambda)$,*

$$(\text{pp}, (r_i, g_i, \pi_F(g_i))_{i \in [m]}) \approx_c (\text{pp}, (r_i, g_i, \tilde{f}_i)_{i \in [m]})$$

where $\text{pp} = (G, H, f, t, \hat{N}) \leftarrow \mathcal{I}.\text{Gen}(1^\lambda)$ and, for each $i \in [m]$, $r_i \xleftarrow{\$} \{0, 1\}^{\ell_r}$, $\tilde{f}_i \xleftarrow{\$} F$, and $g_i := \mathcal{I}.\text{SampleG}^{r_i}(\text{pp})$.

We let the distinguisher see polynomially many samples because our PCFs run $\mathcal{I}.\text{SampleG}$ on many public random strings. Unlike for SgI, whose two distributions are both publicly samplable, the single-sample form of PC-SgI does not imply the multi-sample one.

Whether the PC-SgI assumption holds can depend on the concrete sampling algorithm. In the following, we show that SgI implies PC-SgI for any *invertible* sampling algorithm $\mathcal{I}.\text{SampleG}$ that allows simulating valid randomness r' given the output $g = \mathcal{I}.\text{SampleG}^r(\text{pp})$. However, not all subgroup framework instantiations necessarily admit a sampling algorithm with this property.

Theorem 4.6 (SgI $\Rightarrow$ PC-SgI). *Let $\mathcal{I}$ be a subgroup framework instantiation with a sampling algorithm $\mathcal{I}.\text{SampleG}$, and let $\ell_r \in \text{poly}(\lambda)$ be such that $\mathcal{I}.\text{SampleG}$ uses at most ℓ_r bits of randomness. If there exists a ppt. algorithm $\text{Explain}(\text{pp}, g)$ such that $(r, g) \approx_s (r', g)$, where $r \xleftarrow{\$} \{0, 1\}^{\ell_r}$, $g := \mathcal{I}.\text{SampleG}^r(\text{pp})$, and $r' \leftarrow \text{Explain}(\text{pp}, g)$, then SgI implies PC-SgI (cf. [Definitions 4.4 and 4.5](#)).*

Proof. This follows via a standard hybrid argument:

$$(\text{pp}, (r_i, g_i, \pi_F(g_i))_i) \approx_s (\text{pp}, (r'_i, g_i, \pi_F(g_i))_i) \approx_c (\text{pp}, (r'_i, g_i, \tilde{f}_i)_i) \approx_s (\text{pp}, (r_i, g_i, \tilde{f}_i)_i),$$

where $\tilde{f}_i \xleftarrow{\$} F$, and i ranges over $[m]$. The first and third transitions follow directly from $(r, g) \approx_s (r', g)$. The middle transition relies on the SgI assumption, which can be applied one sample at a time. Crucially, each r'_i can be sampled efficiently given g_i , so the r'_i do not provide the distinguisher with additional information. $\square$

Note that the existence of Explain is a nontrivial requirement. In particular, the “natural” sampling algorithm that samples $\tilde{f} \xleftarrow{\$} F$, $h \xleftarrow{\$} H$, and outputs $g := fh$ does not satisfy this property. If it did, one could efficiently recover $\tilde{f}$ and h from g by computing $\text{Explain}(\text{pp}, g)$, contradicting the SgI assumption.

Theorem 4.7 (SgI for Homomorphic Images). *Let $\mathcal{I}$ be a subgroup framework instantiation with $\text{pp} = (G, H, f, t, \hat{N})$ and $F = \langle f \rangle$. Let $\mathcal{I}'$ be a subgroup framework instantiation whose $\text{Gen}(1^\lambda)$ runs $\text{pp} \leftarrow \mathcal{I}.\text{Gen}(1^\lambda)$ and outputs $\text{pp}' = (G', H', f, t, \hat{N}')$ with $H' \leq H$, where pp and pp' are computable from each other, along with an efficiently computable homomorphism $\sigma: G \rightarrow G'$ satisfying $\sigma(H) = H'$ and $\sigma(F) = F$. Then SgI for $\mathcal{I}$ holds if and only if SgI for $\mathcal{I}'$ holds.*

Proof. Given a distinguisher $\mathcal{D}'$ against $\mathcal{I}'$, we obtain one against $\mathcal{I}$: on input (pp, g) , compute pp' and output $\mathcal{D}'(\text{pp}', \sigma(g))$. The homomorphism σ preserves the uniform distribution over the respective domains, so the advantage is preserved. Conversely, given a distinguisher $\mathcal{D}$ against $\mathcal{I}$, we obtain one against $\mathcal{I}'$: on input (pp', z) , compute pp , sample $h \leftarrow \mathcal{I}.\text{Sample}(H)$ and output $\mathcal{D}(\text{pp}, zh)$. The map $z \mapsto zh$ preserves the projection $\pi_F(z) = \pi_F(zh)$ and randomizes the projection to H , so it again preserves the uniform distribution over the respective domains. $\square$

Theorem 4.8 (PC-SgI for Homomorphic Images). *Let $\mathcal{I}$, $\mathcal{I}'$ and σ be as in [Theorem 4.7](#), where $\mathcal{I}'.\text{SampleG}^r(\text{pp}') := \sigma(\mathcal{I}.\text{SampleG}^r(\text{pp}))$ and additionally $\sigma|_F$ is efficiently invertible. Then PC-SgI for $\mathcal{I}$ holds if and only if PC-SgI for $\mathcal{I}'$ holds.*

Proof. A PC-SgI instance $(\text{pp}, (r_i, g_i, z_i)_i)$ for $\mathcal{I}$ is turned into one for $\mathcal{I}'$ by mapping it to $(\text{pp}', (r_i, \sigma(g_i), \sigma(z_i))_i)$. Conversely, an instance $(\text{pp}', (r_i, g'_i, y_i)_i)$ for $\mathcal{I}'$ is turned into one for $\mathcal{I}$ by mapping it to $(\text{pp}, (r_i, \mathcal{I}.\text{SampleG}^{r_i}(\text{pp}), \sigma|_F^{-1}(y_i))_i)$. $\square$

Theorem 4.9 (WPRF from PC-SgI). *Let $\mathcal{I}$ be a subgroup framework instantiation with an $\mathcal{I}.\text{SampleG}$ algorithm and let $\ell_r \in \text{poly}(\lambda)$ such that $\mathcal{I}.\text{SampleG}$ uses at most ℓ_r bits of randomness. Assume that $\mathcal{I}.\text{Gen}(1^\lambda)$ outputs, in addition to $\text{pp} = (G, H, f, t, \hat{N})$, some trapdoor information k that enables efficient computation of the projection $\pi_F : G \rightarrow F$. Then the WPRF defined by*

$$\text{Gen}(1^\lambda) := \mathcal{I}.\text{Gen}(1^\lambda), \quad \text{Eval}(k, r \in \{0, 1\}^{\ell_r}) := \pi_F(\mathcal{I}.\text{SampleG}^r(\text{pp}))$$

is secure under the PC-SgI assumption for $\mathcal{I}$, even against a distinguisher that is given pp .

Proof. A distinguisher that obtains m WPRF samples $(r_i, \text{Eval}(k, r_i))$ for $r_i \xleftarrow{\$} \{0, 1\}^{\ell_r}$ sees exactly the left-hand distribution of [Definition 4.5](#), up to the components g_i , which it can compute itself from pp and r_i . The claim thus follows directly from the PC-SgI assumption. $\square$

Definition 4.10 (Small Exponent Assumption). *Let $\check{N} = \check{N}(\lambda)$. The small exponent assumption with length $\check{N}$ for $\mathcal{I}$ states that*

$$(\text{pp}, \check{N}, h, h^x) \approx_c (\text{pp}, \check{N}, h, h^y)$$

where $\text{pp} = (G, H, f, t, \hat{N}) \leftarrow \mathcal{I}.\text{Gen}(1^\lambda)$, $h \xleftarrow{\$} H$, $x \xleftarrow{\$} [\hat{N}]$, and $y \xleftarrow{\$} [\check{N}(\lambda)]$.

4.2 Instantiations

Joye–Libert. The following instantiation is a generalized version of what’s used for the Joye–Libert cryptosystem [\[JL13\]](#). In [\[JL13\]](#), it was restricted to t being a power of two, and in [\[ADOS22\]](#), it was generalized to t being a product of distinct small primes. Our definition is even more general, as it allows for arbitrary t , and it will be used with any t that contains no superpolynomial-size prime factors (so that the discrete logarithm is easy and our new DDLog can be used).

Definition 4.11 (Joye–Libert Instantiation). *Let $2 \leq t \in \mathbb{N}$, $\ell \in \text{poly}(\lambda)$, and let $\tau \in \{1, 2\}$, where $\tau = 2$ is admissible only for odd t . We define the subgroup framework instantiation $\text{JL}_{t,\ell}^\tau$:*

- $\text{JL}_{t,\ell}^\tau.\text{Gen}(1^\lambda)$ *proceeds as follows: Choose an RSA semiprime $N = pq$ for random ℓ -bit primes p, q subject to $p \neq q$ and $p = tp' + 1, q = tq' + 1$ where $p', q' \in \mathbb{N}$ are coprime to t ⁷ and $t \mid (p' + q')$.⁸ The group $F = \langle f \rangle$ is generated by a random element $f \in \mathbb{Z}_N^*$ that has order t in both $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$. Further, $H = \{x^{\tau t} \mid x \in \mathbb{Z}_N^*\}$ is the set of τt^{th} powers in $\mathbb{Z}_N^*$ and $G := FH \cong F \times H$. A public upper bound $\hat{N}$ on the order of G is $\hat{N} = \lfloor N/(\tau^2 t) \rfloor$. The output is $(G, H, f, t, \hat{N})$.*
- $\text{JL}_{t,\ell}^\tau.\text{Sample}(H)$ *samples $x \xleftarrow{\$} \mathbb{Z}_N^*$ (e.g., by sampling $x \xleftarrow{\$} \mathbb{Z}_N$ and repeating the process when $\gcd(x, N) \neq 1$) and outputs $x^{\tau t}$.*
- $\text{JL}_{t,\ell}^\tau.\text{SampleG}(\text{pp})$ *is defined in [Section 5](#).*

We also write $\text{JL}_{t,\ell}$ (without superscript) when the choice of τ does not matter.

Compared to [\[JL13, ADOS22\]](#), we add the requirement that $t \mid (p' + q')$. This is crucial to prevent higher-residuosity attacks such as those described in [Section 6](#) and [\[ZCDL25\]](#), in

⁷ In [\[ADOS22\]](#), p', q' are themselves primes, but this is not required to conform with the framework. Our attack in [Section 6](#) also applies to this more general definition.

⁸ Such parameters can be sampled as follows: Sample p' subject to $\gcd(p', t) = 1$ and repeat until $p = tp' + 1$ is prime, then sample $q' \equiv -p' \pmod{t}$ and repeat until $q = tq' + 1$ is prime. This guarantees $t \mid (p' + q')$ and $\gcd(q', t) = 1$. Under the usual heuristic that primes are equidistributed in arithmetic progressions, the expected number of primality tests is no larger than for ordinary prime generation. The element f is obtained by sampling $x \xleftarrow{\$} \mathbb{Z}_p^*$ and repeating until $f_p := x^{p'}$ has order exactly t , which happens with probability $\varphi(t)/t$, proceeding likewise modulo q , and combining f_p, f_q via the CRT.

settings where the element f is made public. The restriction $t \mid (p' + q')$ was suggested as a countermeasure in [ZCDL25], which left a security proof as future work. Section 6.1 supports that this restriction suffices to ensure security in such settings.

Moreover, unlike [JL13, ADOS22], our definition also captures the case where t is odd. Two of the cryptosystems we present in Section 9 require that the group H is cyclic and its order contains only superpolynomial-size prime factors. For odd t , the group H has these properties only for $\tau = 2$, which is why we introduced this parameter. The SgI assumption is equivalent for the two choices by Theorem 4.7 with $\sigma(z) := z^2$.

The following lemma states that H is cyclic, if we sample the RSA semiprime subject to the additional condition that $p'/\tau, q'/\tau$ are coprime. (Note that $\tau \mid p', q'$ is always satisfied, since t odd implies that p', q' are even.)

Lemma 4.12. *Let t, τ, p', q', H be as in Definition 4.11. Then $|H| = p'q'/\tau^2$. Moreover, if $\gcd(p'/\tau, q'/\tau) = 1$, then H is cyclic.*

Proof. H is isomorphic to $\{x^{\tau t} \mid x \in \mathbb{Z}_p^*\} \times \{x^{\tau t} \mid x \in \mathbb{Z}_q^*\}$, hence $|H| = \frac{p-1}{\tau t} \cdot \frac{q-1}{\tau t} = p'q'/\tau^2$. The latter two groups are cyclic, since they are subgroups of the cyclic groups $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$, respectively. Therefore, if their orders are coprime, it follows that H is cyclic as well. $\square$

The order of H is coprime to t , since p'/τ and q'/τ divide p' and q' , respectively, which are coprime to t by definition. Hence, the Joye–Libert instantiation satisfies the condition $\gcd(|H|, t) = 1$ that our cryptosystems in Section 9 and our PCFs in Section 11 require.

We consider the Joye–Libert instantiation to be the most relevant to our work, since it profits from our improved DDLog algorithm (when using a suitably smooth plaintext modulus t) and enables PCFs over certain plaintext moduli that were not supported by [OSY21].

Paillier and Damgård–Jurik. The subgroup framework can also be instantiated with the groups used in the Paillier cryptosystem [Pai99] or Damgård–Jurik cryptosystem [DJ01]. The SgI assumption for these instantiations is normally called DCR [Pai99]. For brevity, we explicitly define only the Paillier instantiation. With this instantiation, our HSS and PCF constructions correspond to those from [OSY21].

Definition 4.13 (Paillier Instantiation). *For $\ell \in \text{poly}(\lambda)$ and $\tau \in \{1, 2\}$, we define the subgroup framework instantiation $\text{Paillier}_\ell^\tau$:*

- $\text{Paillier}_\ell^\tau.\text{Gen}(1^\lambda)$ proceeds as follows: Choose an RSA semiprime $N = pq$ for random ℓ -bit primes p, q subject to $p \neq q$. The group $F = \langle f \rangle$ is generated by $f := 1 + N \in \mathbb{Z}_{N^2}^*$ which has order $t := N$. Define $H = \{x^{\tau t} \mid x \in \mathbb{Z}_{N^2}^*\}$ (which is isomorphic to $\mathbb{Z}_N^*$ for $\tau = 1$) and $G := FH = \{x^\tau \mid x \in \mathbb{Z}_{N^2}^*\} \cong F \times H$. A public upper bound $\hat{N}$ on the order of G is $\hat{N} := \lfloor N^2/\tau^2 \rfloor$. The output is $(G, H, f, t, \hat{N})$.
- $\text{Paillier}_\ell^\tau.\text{Sample}(H)$ samples $x \xleftarrow{\$} \mathbb{Z}_{N^2}^*$ and outputs $x^{\tau N}$.
- $\text{Paillier}_\ell^\tau.\text{SampleG}(\text{pp})$ samples $x \xleftarrow{\$} \mathbb{Z}_{N^2}^*$ and outputs x^τ .

We also write Paillier_ℓ (without superscript) when the choice of τ does not matter.

As mentioned, two of the cryptosystems require that H is cyclic and $|H|$ contains only superpolynomial-size prime factors, and this is what $\tau = 2$ is for here as well. The group H is isomorphic to $\{x^\tau \mid x \in \mathbb{Z}_p^*\} \times \{x^\tau \mid x \in \mathbb{Z}_q^*\}$, whose orders are $(p-1)/\tau$ and $(q-1)/\tau$, so $|H| = \varphi(N)/\tau^2$, and H is cyclic if and only if those two orders are coprime. For $\tau = 1$ they are both even, so H is never cyclic; for $\tau = 2$, choosing p and q as safe primes makes them the primes $(p-1)/2$ and $(q-1)/2$, which are coprime and superpolynomial in size. The SgI assumption is equivalent for the two choices as well, by Theorem 4.7 with $\sigma(z) := z^2$. By Theorems 4.6 and 4.8, SgI implies PC-SgI for both. Neither does the choice of τ degrade the security of any of the other assumptions defined in Section 4.1, nor does it affect the

correctness of the DDLog from [OSY21]. Therefore, all of our constructions work for either choice. As noted in Remark 11.17, we observe that $\tau = 2$ even allows us to avoid the QR assumption in the non-interactive VOLE (NIVOLE) protocol from [OSY21].

The Paillier instantiation satisfies $\gcd(|H|, t) = 1$ as well, for either choice of τ : Here $t = N$ and $|H|$ divides $\varphi(N) = (p-1)(q-1)$, so it suffices that $\gcd(N, \varphi(N)) = 1$. The latter holds because $p \nmid (p-1)$, and because $0 < q-1 < 2p$ for ℓ -bit primes p, q leaves $q-1 = p$ as the only way for p to divide $q-1$, which cannot happen. The same argument applies with p and q swapped.

Class Groups. As discussed in [ADOS22], the subgroup framework can also be instantiated based on class groups of unknown order, following the framework of Castagnos and Laguillaumie [CL15]. Here, G is the class group of an imaginary quadratic order, F is generated by an explicit ideal class of prime order t in which the discrete logarithm is easy, and H has the order of the class group of the maximal order, which is unknown. From this instantiation, we can obtain HSS, but our PCFs are incompatible with class groups, since the PCFs require the group order to be known to one of the two parties.

5 Public-Coin Sampling for Joye–Libert

In this section, we define our sampling algorithm $\text{JL}_{t,\ell}^\tau.\text{SampleG}$ for sampling a uniformly random element $g \in G$ in the Joye–Libert instantiation. For $\tau = 1$, the randomness can be reproduced given the output, so SgI implies PC-SgI by Theorem 4.6; for $\tau = 2$, we obtain PC-SgI under the same assumption via Theorems 4.7 and 4.8.

5.1 Number-Theoretic Preliminaries

We refer to [Neu99] for the algebraic number theory used in this section. For $m \in \mathbb{N}_{\geq 1}$, the cyclotomic ring of integers $\mathcal{R} = \mathbb{Z}[\zeta_m] \cong \mathbb{Z}[X]/(\Phi_m)$ is always a Dedekind domain, i.e., every nonzero proper ideal of $\mathcal{R}$ can be factored uniquely (up to order) into prime ideals. We denote by $(a_1, \dots, a_n)$, for $a_i \in \mathcal{R}$, the ideal $\{r_1 a_1 + \dots + r_n a_n \mid r_i \in \mathcal{R}\}$. The norm of an ideal $\mathfrak{a} \triangleleft \mathcal{R}$ is defined by $\mathcal{N}(\mathfrak{a}) := |\mathcal{R}/\mathfrak{a}|$ and is multiplicative.

Over $\mathcal{R} = \mathbb{Z}[\zeta_m]$, one can define the m^{th} power residue symbol, a generalization of the Jacobi symbol: Let $\mathfrak{p} \triangleleft \mathcal{R}$ be a prime ideal coprime to m (it follows that $\mathcal{N}(\mathfrak{p}) \equiv 1 \pmod{m}$) and $\alpha \in \mathcal{R}$. If $\alpha \notin \mathfrak{p}$, then there is a unique m^{th} root of unity ζ_m^j such that $\zeta_m^j \equiv \alpha^{\frac{\mathcal{N}(\mathfrak{p})-1}{m}} \pmod{\mathfrak{p}}$. This defines the m^{th} power residue symbol

$$\left(\frac{\alpha}{\mathfrak{p}}\right)_m := \begin{cases} 0 & \text{if } \alpha \in \mathfrak{p}, \\ \zeta_m^j \equiv \alpha^{\frac{\mathcal{N}(\mathfrak{p})-1}{m}} \pmod{\mathfrak{p}} & \text{if } \alpha \notin \mathfrak{p}. \end{cases}$$

Similarly to the Jacobi symbol, this definition can be extended to composite ideals $\mathfrak{a} \triangleleft \mathcal{R}$ that are coprime to m : For such an $\mathfrak{a}$ with prime ideal factorization $\mathfrak{a} = \prod_i \mathfrak{p}_i$, the m^{th} power residue symbol is defined by

$$\left(\frac{\alpha}{\mathfrak{a}}\right)_m := \prod_i \left(\frac{\alpha}{\mathfrak{p}_i}\right)_m \in \langle \zeta_m \rangle \cup \{0\}.$$

For $m = 2$, this is precisely the Jacobi symbol, and for $m > 2$, it is a natural generalization thereof. The m^{th} power residue symbol is multiplicative, i.e., $\left(\frac{\alpha\beta}{\mathfrak{a}}\right)_m = \left(\frac{\alpha}{\mathfrak{a}}\right)_m \cdot \left(\frac{\beta}{\mathfrak{a}}\right)_m$, and for $\alpha \equiv \alpha' \pmod{\mathfrak{a}}$ it holds that $\left(\frac{\alpha}{\mathfrak{a}}\right)_m = \left(\frac{\alpha'}{\mathfrak{a}}\right)_m$.

Table 5.1. Efficient algorithms for computing the t^{th} power residue symbol

$t = 3$	$t = 4$	$t = 5$	$t = 7$	$t = 8$
[Wil86, SW95, DF05]	[Wei02]	[SW95]	[CS10]	[Joy19]
$t = 9$	$t = 11$	$t = 13$	$t = 16$	$t \in \{17, 19, 25, 27, 32\}$
[BGSJN21]	[JLNN21]	[BN19]	[BGSJN21]	Appendix A

5.2 Our Sampling Algorithm

Our sampling algorithm requires the existence of an efficient algorithm to compute the t^{th} power residue symbol $\left(\frac{\cdot}{\mathfrak{a}}\right)_t$ in $\mathbb{Z}[\zeta_t]$ *without knowing the factorization of $\mathfrak{a}$* . Efficient algorithms are known for the prime powers $t \in \{3, 4, 5, 7, 8, 9, 11, 13, 16\}$ (cf. Table 5.1) and for $t = 2$, where the symbol is the Jacobi symbol. In Appendix A, we extend the approach of [BGSJN21] to $t \in \{17, 19, 25, 27, 32\}$. Composite t reduce to their prime-power factors, as we show below. Computing the power residue symbol becomes increasingly challenging as t increases. The algorithms make progress only when a remainder of smaller norm is found in $\mathbb{Z}[\zeta_t]$, which exists for the norm-Euclidean rings with $t \leq 16$ but not always for $t = 32$, and for $t \in \{17, 19, 25, 27\}$ it is unknown whether it always exists [Lem95, Sect. 8]; for these t , our algorithm provably terminates with the correct output, while its running time remains heuristic. For every other prime power t , the class group of $\mathbb{Z}[\zeta_t]$ is nontrivial [MM76], so the ideal ρ_t defined below need not be principal and the approach does not apply. The work [dBP17] presents a probabilistic algorithm for general t , but it has not been proven to run in polynomial time.

Our sampling algorithm relies on the following number-theoretic observations. Let $m \mid t$ with $m \geq 3$. As observed in [ZCDL25], knowledge of f reveals a partial factorization of (N) over $\mathcal{R} := \mathbb{Z}[\zeta_m]$. To see how (N) decomposes over $\mathbb{Z}[\zeta_m]$, we need some number theory: Since $p \equiv 1 \pmod{m}$ (recall $p = tp' + 1$ and $m \mid t$), it follows that (p) factors over $\mathbb{Z}[\zeta_m]$ into prime ideals as follows: $(p) = \prod_{i \in \mathbb{Z}_m^*} \mathfrak{p}_i$ with $\mathfrak{p}_i = (p, \zeta_m - \psi^i)$, where $\psi \in \mathbb{Z}$ is any primitive m^{th} root of unity modulo p , and $\mathcal{N}(\mathfrak{p}_i) = p$. We fix $\psi = f^{t/m} \pmod{N}$. Since $q \equiv 1 \pmod{m}$ and ψ is also a primitive m^{th} root of unity modulo q , we likewise get the factorization $(q) = \prod_{i \in \mathbb{Z}_m^*} \mathfrak{q}_i$ with $\mathfrak{q}_i = (q, \zeta_m - \psi^i)$ and $\mathcal{N}(\mathfrak{q}_i) = q$. It follows that $\mathfrak{p}_1 \mathfrak{q}_1 = (p, \zeta_m - \psi)(q, \zeta_m - \psi) = (N, \zeta_m - \psi)$, where the latter equality uses $\gcd(p, q) = 1$. That is, knowledge of f immediately reveals the factor $\rho_m := \mathfrak{p}_1 \mathfrak{q}_1 = (N, \zeta_m - f^{t/m})$ of (N) . Since $\mathbb{Z}[\zeta_m]$ has class number one for every prime power m in Table 5.1 [MM76], ρ_m is principal. The algorithms listed there take a generator of ρ_m as input, which the Euclidean algorithm computes as $\gcd(N, \zeta_m - f^{t/m})$ without factoring N .

Throughout this section, G^τ, H^τ, F^τ denote the groups of $\mathbf{JL}_{t,\ell}^\tau$ (cf. Definition 4.11); note that $F^1 = F^2$ and $H^2 \subseteq H^1 = \{x^t \mid x \in \mathbb{Z}_N^*\}$. It turns out that G^1 contains exactly the elements $g \in \mathbb{Z}_N^*$ with $\left(\frac{g}{\rho_t}\right)_t = 1$. Toward proving this, we establish the following lemma:

Lemma 5.2. *Let m, ρ_m be as above and $g \in G^1$, then $\left(\frac{g}{\rho_m}\right)_m = 1$.*

Proof. By definition of the m^{th} power residue symbol, we have $\left(\frac{g}{\rho_m}\right)_m = \left(\frac{g}{\mathfrak{p}_1}\right)_m \cdot \left(\frac{g}{\mathfrak{q}_1}\right)_m$. Write $g = f^k h$ for some $k \in \mathbb{Z}_t$ and $h \in H^1$. By definition of H^1 , there exists $x \in \mathbb{Z}_N^*$ such that $h = x^t$. It holds that

$$\begin{aligned} \left(\frac{h}{\mathfrak{p}_1}\right)_m &\equiv h^{\frac{\mathcal{N}(\mathfrak{p}_1)-1}{m}} = \left(x^{\frac{t}{m}}\right)^{\mathcal{N}(\mathfrak{p}_1)-1} \stackrel{(i)}{\equiv} 1 \pmod{\mathfrak{p}_1}, \\ \left(\frac{f}{\mathfrak{p}_1}\right)_m &\equiv f^{\frac{\mathcal{N}(\mathfrak{p}_1)-1}{m}} = f^{\frac{tp'}{m}} = \psi^{p'} \stackrel{(ii)}{\equiv} \zeta_m^{p'} \pmod{\mathfrak{p}_1}, \end{aligned}$$

since (i) $\mathbb{Z}[\zeta_m]/\mathfrak{p}_1$ is a finite field of size $\mathcal{N}(\mathfrak{p}_1)$ and (ii) $\zeta_m - \psi \in \mathfrak{p}_1$. By the multiplicativity of the power residue symbol, it follows that $\left(\frac{g}{\mathfrak{p}_1}\right) = \zeta_m^{p_1^k}$. Analogously, we obtain $\left(\frac{g}{\mathfrak{q}_1}\right) = \zeta_m^{q_1^k}$, hence $\left(\frac{g}{\rho_m}\right) = \zeta_m^{(p_1' + q_1')k \bmod m}$. Since $m \mid t \mid (p_1' + q_1')$, it follows that $\left(\frac{g}{\rho_m}\right) = 1$. $\square$

The symbol $\left(\frac{g}{\rho_t}\right)_t$ can take t distinct values, all of which occur equally often over $g \in \mathbb{Z}_N^*$. Since G^1 contains a $1/t$ fraction of the elements from $\mathbb{Z}_N^*$, and all $g \in G^1$ satisfy $\left(\frac{g}{\rho_t}\right)_t = 1$, we get the following corollary. This also holds when $t = 2$, where $\rho_2 := N$ and $\left(\frac{g}{\rho_2}\right)_2$ coincides with the Jacobi symbol.

Corollary 5.3. *Let ρ_t be as above, then $G^1 = \left\{g \in \mathbb{Z}_N^* \mid \left(\frac{g}{\rho_t}\right)_t = 1\right\}$.*

The Sampling Algorithm. The above corollary allows us to define $\text{JL}_{t,\ell}^\tau.\text{SampleG}(\text{pp})$. Let $e(x) := \log_{\zeta_t} \left(\frac{x}{\rho_t}\right)_t \in \mathbb{Z}_t$ for $x \in \mathbb{Z}_N^*$. Let $y \in \mathbb{Z}_N^*$ be the smallest element with $\gcd(e(y), t) = 1$, and let $x_1 := y^{e(y)^{-1} \bmod t}$, so that $e(x_1) = 1$ by the multiplicativity of the power residue symbol. (Fixing y this way ensures that both parties use the same x_1 .)

The algorithm proceeds as follows: Sample $x \xleftarrow{\$} \mathbb{Z}_N^*$, let $u := x \cdot x_1^{-e(x)}$, and output $g := u^\tau$. The algorithm computes one power residue symbol per sample, whereas rejection sampling from $\mathbb{Z}_N^*$ would need t in expectation. By [Corollary 5.6](#) below, SgI for $\text{JL}_{t,\ell}^\tau$ implies PC-SgI for this sampling algorithm. Of course, this sampling algorithm only works for values of t where an efficient algorithm for computing the t^{th} power residue symbol exists (cf. [Table 5.1](#)).

For composite t , it suffices to compute the m^{th} power residue symbols for the prime powers m in the prime-power factorization of t and recover $e(x)$ via the CRT, since $\left(\frac{x}{\rho_t}\right)_t^{t/m} = \left(\frac{x}{\rho_m}\right)_m$ for $m \mid t$. Therefore, the PCFs we construct in [Section 11](#) are currently constrained to $t \mid 6983776800 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$ when using the JL instantiation.

Lemma 5.4. *The intermediate value u computed by $\text{JL}_{t,\ell}^\tau.\text{SampleG}(\text{pp})$ is uniform over G^1 , and for every $u \in G^1$, the values of x for which the algorithm computes u are exactly ux_1^c for $c \in [t]$. Consequently, the output $g = u^\tau$ is uniform over G^τ .*

Proof. By the multiplicativity of the power residue symbol and $e(x_1) = 1$, we have $e(u) = e(x) - e(x) = 0$, hence $u \in G^1$ by [Corollary 5.3](#). For $u \in G^1$ and $c \in [t]$, the algorithm on input $x = ux_1^c$ computes $e(x) = e(u) + c = c$ and thus computes u . Conversely, every $x \in \mathbb{Z}_N^*$ is of the form ux_1^c for the intermediate value u and $c = e(x)$. The t preimages ux_1^c for $c \in [t]$ are pairwise distinct, hence u is uniform over G^1 . Finally, $u \mapsto u^\tau$ maps $G^1 = F^1 H^1$ onto $F^\tau H^\tau = G^\tau$, so the output is uniform over G^τ . $\square$

Lemma 5.5. *The sampling algorithm $\text{JL}_{t,\ell}^1.\text{SampleG}$ satisfies the condition of [Theorem 4.6](#), i.e., there exists a ppt. algorithm $\text{Explain}(\text{pp}, g)$ for reverse-sampling the randomness r that explains g .*

Proof. $\text{Explain}(\text{pp}, g)$ proceeds as follows: Sample $c \xleftarrow{\$} [t]$ and $r \xleftarrow{\$} \{0, 1\}^{\ell_r}$, and simulate $\text{JL}_{t,\ell}^1.\text{SampleG}^r(\text{pp})$, obtaining the intermediate value x . In r , replace the bit representation of x (i.e., the substring of r that was accessed in the iteration where the sampling of x terminated) by the bit representation of gx_1^c and output the result. By [Lemma 5.4](#) with $\tau = 1$, where the output g equals the intermediate value u , the element gx_1^c is uniform over the values of x for which the algorithm outputs g , so the output of Explain is distributed like r conditioned on $\text{JL}_{t,\ell}^1.\text{SampleG}^r(\text{pp}) = g$. $\square$

Corollary 5.6. *The SgI assumption for $\text{JL}_{t,\ell}^\tau$ implies the PC-SgI assumption for $\text{JL}_{t,\ell}^\tau$.*

Proof. For $\tau = 1$, the claim follows from [Theorem 4.6](#) and [Lemma 5.5](#). For $\tau = 2$, SgI for $\text{JL}_{t,\ell}^2$ implies SgI for $\text{JL}_{t,\ell}^1$ by [Theorem 4.7](#), so the previous case yields PC-SgI for $\text{JL}_{t,\ell}^1$, which [Theorem 4.8](#) transfers to $\text{JL}_{t,\ell}^2$. $\square$

6 Attacks Based on the Power Residue Symbol

In the proof of [Lemma 5.2](#), we observed that the condition $t \mid (p' + q')$ causes all elements from G to have the same power residue symbol. This is actually a necessary condition to prevent attacks. Namely, when allowing other values of t , there are attacks against DDH and SgI. Here, we present only the attack against DDH, while the attack on SgI proceeds analogously to the attack (on a slightly different problem) described in [\[ZCDL25\]](#). Both attacks specifically exploit knowledge of f and do not work in scenarios where no such element is known, since it is unknown whether such an element can be efficiently computed.

Our attack on DDH breaks the DDH-based cryptosystem from [\[ADOS22\]](#) when using their Joye–Libert instantiation, instead of our fixed version. The Joye–Libert instantiation from [\[ADOS22\]](#) – here denoted $\widetilde{\text{JL}}$ – is defined as follows.

Definition 6.1 (*$\widetilde{\text{JL}}$ Instantiation*). *Let $3 \leq t \in \mathbb{N}$ and $\ell \in \text{poly}(\lambda)$. The subgroup framework instantiation $\widetilde{\text{JL}}_{t,\ell}$ is defined exactly like $\text{JL}_{t,\ell}$ (cf. [Definition 4.11](#)), except that we remove the requirement $t \mid (p' + q')$. Moreover, we add the ppt. algorithm $\widetilde{\text{JL}}_{t,\ell}.\text{SampleDDH}(\text{pp})$ that samples $h \leftarrow \widetilde{\text{JL}}_{t,\ell}.\text{Sample}(H)$ and outputs $g := hf$.*

Our attack on DDH in $\widetilde{\text{JL}}$ assumes the existence of a divisor $m \mid t$ with $m \notin \{1, 2, 4\}$ for which there is an efficient algorithm to compute the m^{th} power residue symbol $\left(\frac{\cdot}{\rho}\right)_m$ in $\mathbb{Z}[\zeta_m]$. Recall from [Table 5.1](#) the set of currently feasible values for m . In [\[ADOS22\]](#), t is the product of the first k primes for some $k \geq 2$, hence divisible by 3, so one can choose $m = 3$ for the attack.

The last argument in the proof of [Lemma 5.2](#) no longer holds when $t \nmid (p' + q')$, so we instead obtain

$$\left(\frac{g}{\rho}\right)_m = \zeta_m^{(p'+q')k \bmod m} \quad (6.2)$$

for $g \in f^k H$. Note that any $g \in G = FH$ lies in $f^k H$ for some unique $k \in \mathbb{Z}_t$. We define $E(g) := (p' + q')k \bmod m$ as shorthand for the exponent from [Eq. \(6.2\)](#). E is efficiently computable by first computing $\left(\frac{g}{\rho}\right)_m$ and then solving the discrete logarithm with base ζ_m . This enables the following attack:

Attack on DDH. An attacker on the DDH problem (cf. [Definition 4.3](#)) needs to distinguish between $(\text{pp}, g, g^x, g^y, g^{xy})$ and $(\text{pp}, g, g^x, g^y, g^z)$, where $g \leftarrow \mathcal{I}.\text{SampleDDH}(\text{pp})$ and $x, y, z \xleftarrow{\$} [\hat{N}]$. Our attacker, on input $(\text{pp}, g, g^x, g^y, g^z)$, outputs 1 if $E(g^x)E(g^y) \equiv E(g)E(g^z) \pmod{m}$ and 0 otherwise.

Theorem 6.3. *Let $m \mid t$ with $m \notin \{1, 2, 4\}$, and assume that $p' \bmod m$ and $q' \bmod m$ are uniform over $\mathbb{Z}_m^*$ and independent.⁹ Then above attacker against DDH in the $\widetilde{\text{JL}}$ instantiation has advantage $\geq 1/4 - m/\hat{N}$.*

⁹ This assumption is heuristic and can be justified as follows. By the prime number theorem for arithmetic progressions, the primes reduced modulo t^2 are asymptotically uniform over $\mathbb{Z}_{t^2}^*$. Since p is sampled subject to $t \mid (p-1)$ and $\gcd((p-1)/t, t) = 1$, the value $p \bmod t^2$ is close to uniform over the classes $at+1$ with $\gcd(a, t) = 1$, and $a = p' \bmod t$, so $p' \bmod t$ is close to uniform over $\mathbb{Z}_t^*$, and likewise $p' \bmod m$ over $\mathbb{Z}_m^*$ for any $m \mid t$.

Proof. Sample $g \leftarrow \widetilde{\text{JL.SampleDDH}}(\text{pp})$. By [Definition 6.1](#), we have $g \in fH$, hence $g^x \in f^x H$ and similarly for g^y, g^z . It follows that $E(g^x)E(g^y) \equiv_m (p' + q')^2 xy \equiv_m E(g)E(g^{xy})$. Hence, the attacker outputs 1 if and only if $(p' + q')^2 xy \equiv_m (p' + q')^2 z$. When $z = xy$, this is clearly satisfied, so the attacker outputs 1. Therefore, the advantage is $\Pr[(p' + q')^2(xy - z) \not\equiv_m 0]$ where z is sampled uniformly at random from $[\hat{N}]$. Write $a := (p' + q')^2 \bmod m$ and $w := xy - z \bmod m$, so that the advantage is $\Pr[aw \not\equiv_m 0]$. Since z is uniform over $[\hat{N}]$, the value w is uniform over $\mathbb{Z}_m$ up to a statistical distance of $m/\hat{N}$, and for uniform w it holds that $\Pr[aw \equiv_m 0] = \gcd(a, m)/m$. Let $\varphi(\cdot)$ denote Euler's totient function, let $m = p_1^{e_1} \cdots p_s^{e_s}$ with $p_1 < \cdots < p_s$ be the prime power factorization of m , and define $m' := p_1^{\lceil e_1/2 \rceil} \cdots p_s^{\lceil e_s/2 \rceil}$. Then $a \equiv_m 0$ if and only if $p' + q' \equiv_{m'} 0$, which by assumption happens with probability $\frac{1}{\varphi(m')}$. Whenever $a \not\equiv_m 0$, the divisor $\gcd(a, m)$ of m is proper, hence $\gcd(a, m) \leq m/p_1$ and thus $\Pr_w[aw \not\equiv_m 0] \geq 1 - \frac{1}{p_1} \geq \frac{1}{2}$. Since $m \notin \{1, 2, 4\}$ implies $\varphi(m') \geq 2$, the advantage is at least $\left(1 - \frac{1}{\varphi(m')}\right) \left(1 - \frac{1}{p_1}\right) - \frac{m}{\hat{N}} \geq \frac{1}{4} - \frac{m}{\hat{N}}$. (For the excluded cases $m = 2, 4$, the attack does not work due to $m' = 2$ leading to $\varphi(m') = 1$.) This bound is tight: for $m = 8$, we have $m' = 4$ and $p_1 = 2$, so both factors equal $\frac{1}{2}$. $\square$

6.1 Countermeasures

In [\[ZCDL25\]](#), the authors discuss several possible countermeasures against their higher residuosity attack, two of which can be applied to our setting:

- The first option is to choose the modulus t to have no small prime factors m for which computing the m^{th} power residue symbol is feasible. We note that this approach has several downsides. First, the lack of efficient algorithms for the m^{th} power residue symbol for the prime powers m missing from [Table 5.1](#) seems to be a technical problem, that may well be solved in the future. Second, when instantiating cryptosystems in the subgroup framework, the modulus t is often determined by the application and cannot be chosen freely. Third, even if t can be chosen freely, very smooth t are preferable in practice, since the runtime of the discrete logarithm (or DDLog) increases with the size of the prime factors of t (cf. [Theorem 7.3](#)).
- The second option is to sample the RSA semiprime subject to an additional requirement that prevents the attack. In our notation, this requirement is $t \mid (p' + q')$ and is already incorporated in our definition of the JL instantiation ([Definition 4.11](#)). Using [Lemma 5.2](#), it is easy to see that this leads to all group elements having the same m^{th} power residue symbol (for all $m \mid t$), thereby preventing the attack.

We conjecture that $t \mid (p' + q')$ effectively not only prevents the specific attacks from [Section 6](#) and [\[ZCDL25\]](#), but in general prevents attacks that exploit knowledge of f in the Joye–Libert instantiation. As noted above, the requirement $t \mid (p' + q')$ corresponds to working in a group where all elements have the t^{th} power residue symbol 1. Hence, the ability to compute the power residue symbol is unlikely to be useful to an adversary.

There could still be attacks that exploit knowledge of f using techniques other than computing residue symbols. Currently, no such attacks are known. This situation is the same as for QR, where attacks based on computing the Jacobi symbol are ruled out by working in $\mathbb{J}_N$ (i.e., the group of elements with Jacobi symbol 1), and no other attacks are known.

7 New DDLog Algorithm

We present our new DDLog algorithm in [Algorithm 7.1](#), followed by its correctness proof and asymptotic analysis. The algorithm is correct for any t , but runs in polynomial time only when t has no superpolynomial-size prime factor.

Algorithm 7.1: Distributed Discrete Logarithm

DDLog(G, f, t, a):

- If t is prime:
 1. Output $\arg \min_{x \in [t]} (a \cdot f^{-x})$.
- Else if t is a prime power $t = p^e$:
 1. Compute $t_L := p^{\lceil e/2 \rceil}$ and $t_U := p^{\lfloor e/2 \rfloor}$.
 2. Compute $x' := \text{DDLog}(G, f^{t_U}, t_L, a^{t_U})$.
 3. Compute $\hat{a} := a \cdot f^{-x'}$.
 4. Compute $\hat{x} := \text{DDLog}(G, f^{t_L}, t_U, \hat{a})$.
 5. Output $t_L \hat{x} + x'$.
- Else:
 1. Let $t = p_1^{e_1} \cdots p_s^{e_s}$ be the prime power factorization of t .
 2. Compute $t_L := p_1^{e_1} \cdots p_{\lceil s/2 \rceil}^{e_{\lceil s/2 \rceil}}$ and $t_U := p_{\lceil s/2 \rceil + 1}^{e_{\lceil s/2 \rceil + 1}} \cdots p_s^{e_s}$.
 3. Compute $x' := \text{DDLog}(G, f^{t_U}, t_L, a^{t_U})$.
 4. Compute $\hat{x} := \text{DDLog}(G, f^{t_L}, t_U, a^{t_L})$.
 5. Output $x \in [t]$ with $x \equiv x' \pmod{t_L}$ and $x \equiv \hat{x} \pmod{t_U}$ via CRT.

Theorem 7.2. Let G, f, t be as in [Section 4](#), then the algorithm $\text{DDLog}(G, f, t, \cdot)$ from [Algorithm 7.1](#) satisfies [Definition 4.2](#), i.e., for all $x \in [t]$ and $a_0, a_1 \in G$ with $a_1 = a_0 f^x$, it holds that

$$\text{DDLog}(G, f, t, a_1) - \text{DDLog}(G, f, t, a_0) \equiv x \pmod{t}.$$

Proof. We prove the statement by induction over the number of prime factors of t , counted with multiplicity. Note that $t \geq 2$ has at least one prime factor. Let $x_\sigma := \text{DDLog}(G, f, t, a_\sigma)$ for $\sigma \in \{0, 1\}$.

- *Case 1 (base case):* t is prime. Let $X_\sigma := \{a_\sigma \cdot f^{-x} \mid x \in [t]\}$. Observe that $X_1 = X_0$. Therefore $a_1 \cdot f^{-x_1} = \min(X_1) = \min(X_0) = a_0 \cdot f^{-x_0}$, hence $a_1 = a_0 \cdot f^{x_1 - x_0}$, hence $x_1 - x_0 \equiv_t x$.
- *Case 2:* $t = p^e$ is a prime power with $e \geq 2$. Let $x'_\sigma, \hat{a}_\sigma, \hat{x}_\sigma$ be the values computed by party σ in steps 2, 3, 4, respectively. Let $\omega = f^{t_U}$. Observe $a_1^{t_U} = a_0^{t_U} \cdot \omega^x$, so the induction hypothesis gives us $x'_1 - x'_0 \equiv_{t_L} x$. Hence, there exists $k \in \mathbb{Z}$ such that $x'_1 - x'_0 + kt_L = x$. Therefore

$$\hat{a}_1 = a_1 \cdot f^{-x'_1} = a_0 \cdot f^{x - x'_1} = \hat{a}_0 \cdot f^{x + x'_0 - x'_1} = \hat{a}_0 \cdot f^{kt_L} = \hat{a}_0 \cdot (f^{t_L})^k.$$

The induction hypothesis then gives us $\hat{x}_1 - \hat{x}_0 \equiv_{t_U} k$, hence $t_L \hat{x}_1 - t_L \hat{x}_0 \equiv_t kt_L$. It follows that $x_1 - x_0 = t_L \hat{x}_1 + x'_1 - t_L \hat{x}_0 - x'_0 \equiv_t kt_L + x'_1 - x'_0 = x$.

- *Case 3:* $t = p_1^{e_1} \cdots p_s^{e_s}$ (prime-power factorization) with $s \geq 2$. Let $x'_\sigma, \hat{x}_\sigma$ now be the values computed by party σ in steps 3, 4, respectively. By a similar observation as in case 2, the induction hypothesis gives us $x'_1 - x'_0 \equiv_{t_L} x$ and $\hat{x}_1 - \hat{x}_0 \equiv_{t_U} x$. Then $x_1 - x_0 \equiv_t x$ follows by the CRT. $\square$

Theorem 7.3. DDLog from [Algorithm 7.1](#) performs $\mathcal{O}(\log t \log \log t + \sum p_i e_i)$ group operations, where the sum runs over all distinct prime divisors p_i of $t = p_1^{e_1} \cdots p_s^{e_s}$. We also obtain the bound $\mathcal{O}(\log t (\log \log t + p))$ group operations, where p is the largest prime divisor of t .

For sufficiently smooth t , our algorithm is asymptotically better than the complexity $\mathcal{O}(\log^2 t / \log \log t + \sum p_i)$ of the DDLog from [\[ADOS22\]](#), which performs $s \in \mathcal{O}(\log t / \log \log t)$ exponentiations with $\mathcal{O}(\log t)$ -bit exponents, one per prime factor. Concretely, when choosing $t \approx 2^{384}$ as a product of small distinct primes, our DDLog outperforms [\[ADOS22\]](#) by $\approx 2 \times$

due to our divide-and-conquer approach. Furthermore, by exploiting our algorithm's support for a power-of-two modulus $t = 2^{384}$, the advantage over [ADOS22] increases to $\approx 16\times$.

Proof of Theorem 7.3. Let t be the cyclic group order at the root node of the recursion tree. In the non-prime-power case, since t has $s \leq \log t$ distinct prime factors, there are $\mathcal{O}(\log \log t)$ levels of recursion until reaching a prime-power case. Since $e \leq \log t$, there are $\mathcal{O}(\log \log t)$ further levels of recursion until reaching a prime case. The local computation at an inner node with cyclic group order t' takes $\mathcal{O}(\log t')$ group operations (using binary exponentiation). At every level of the recursion tree, the product of these orders is equal to t , so there are $\mathcal{O}(\log t)$ group operations required for every level of the recursion tree, when ignoring the leaf nodes (i.e., the base cases). There are e_i leaf nodes for every distinct prime p_i dividing t , hence the leaf nodes require $\mathcal{O}(\sum p_i e_i)$ group operations in total. $\square$

8 Generic HSS Template

In this section, we describe our generic HSS template, which closely follows the HSS constructions from [OSY21, ADOS22]. Our template is more general, as it can be instantiated with any cryptosystem that has certain properties such as, most notably, a distributed decryption. Moreover, in our template, the evaluation key is smaller than in [OSY21], since we share the secret key directly without digit decomposition. When instantiating our template with the DDH-based cryptosystem from [ADOS22], the resulting HSS scheme is identical to the one from [ADOS22]. We first introduce the required preliminaries and then present the HSS template itself.

8.1 Subtractive Secret Sharing

We use the following notation to distinguish between secret shares over different rings: $\llbracket \cdot \rrbracket^{(t)}$ denotes subtractive shares over $\mathbb{Z}_t$ (i.e., $\llbracket x \rrbracket_1^{(t)} - \llbracket x \rrbracket_0^{(t)} \bmod t = x$), while $\llbracket \cdot \rrbracket^{(\mathbb{Z})}$ denotes subtractive shares *over the integers* (i.e., $\llbracket x \rrbracket_1^{(\mathbb{Z})} - \llbracket x \rrbracket_0^{(\mathbb{Z})} = x$ without reduction modulo any modulus). For a secret x within a bounded interval $[B]$, one can sample statistically hiding shares over the integers as follows: Sample $\llbracket x \rrbracket_0^{(\mathbb{Z})} \xleftarrow{\$} [B2^\kappa]$ (for statistical security parameter κ) and compute $\llbracket x \rrbracket_1^{(\mathbb{Z})} := \llbracket x \rrbracket_0^{(\mathbb{Z})} + x$ to complete the subtractive sharing. We use the notation $\llbracket \mathbf{x} \rrbracket^{(\mathbb{Z}^m)}$ when $\mathbf{x}$ is a length- m vector.

8.2 Distributed Decryption

As usual [BGI16a, BKS19, OSY21, RS21, ADOS22], our HSS template requires a cryptosystem that allows for a two-party *distributed decryption*, i.e., two parties holding a ciphertext $\text{Enc}(x)$ and subtractive shares (over the integers) of the secret key can locally compute subtractive shares of the plaintext x . Formally, we define distributed decryption as follows:

Definition 8.1 (Distributed Decryption). A public-key cryptosystem $\mathcal{S}$ with plaintext space $\mathbb{Z}_t$ is said to have a (two-party) distributed decryption, when:

- The secret key has the form $\text{sk} \in \mathbb{Z}^m$ for some $m \in \mathbb{N}$.
- There is a deterministic algorithm $\mathcal{S}.\text{DDec}$ such that, for all ciphertexts ct , $y \in \mathbb{Z}$ and subtractive shares $\llbracket y \cdot \text{sk} \rrbracket^{(\mathbb{Z}^m)}$ of $y \cdot \text{sk}$, it holds that

$$\mathcal{S}.\text{DDec}(\llbracket y \cdot \text{sk} \rrbracket_1^{(\mathbb{Z}^m)}, \text{ct}) - \mathcal{S}.\text{DDec}(\llbracket y \cdot \text{sk} \rrbracket_0^{(\mathbb{Z}^m)}, \text{ct}) \equiv \mathcal{S}.\text{Dec}(\text{sk}, \text{ct}) \cdot y \pmod{t}.$$

Note that the definition allows for inserting a multiplicative factor y during decryption, i.e., when decrypting with subtractive shares of $y \cdot \text{sk}$, the parties obtain subtractive shares of the plaintext times $y \pmod{t}$.

When there exists an efficient DDLog for the utilized instantiation of the subgroup framework, then a distributed decryption is possible for all three cryptosystems presented in [Section 9](#) and, more generally, for all cryptosystems whose decryption formula is of the form $\text{Dec}(\text{sk}, \text{ct} = (c_0, \dots, c_{k-1})) = \log_f(c_0^{e_0} \cdots c_{k-1}^{e_{k-1}})$, where $e_0, \dots, e_{k-1}$ are linear combinations of the secret key's components and the constant 1. (All three cryptosystems presented in [Section 9](#) are of this form.) This works as follows: Using the linearity of subtractive shares, the parties compute subtractive shares $\llbracket e_i \rrbracket$ of e_i (for $i \in [k]$) and then $a_\sigma := \prod_{i=0}^{k-1} c_i^{\llbracket e_i \rrbracket_\sigma}$. Observe that a_0, a_1 are divisive shares of $c_0^{e_0} \cdots c_{k-1}^{e_{k-1}}$, so the parties can apply the DDLog to obtain subtractive shares of the plaintext.

8.3 RMS Programs

An RMS program is an arithmetic circuit where multiplications are restricted in the way that one of the two operands must be an input. The corresponding function class includes the functions computed by polynomial-size branching programs [\[BGI16a\]](#), which in turn includes NC^1 [\[Bar86\]](#). Notation-wise, we use the RMS program definition from [\[ADOS22\]](#), copied almost verbatim:

Definition 8.2 (RMS Program [\[ADOS22\]](#)). *An RMS program consists of a bound B_{msg} and a circuit with only the following gate types:*

- $\text{ConvertInput}(l_x) \rightarrow M_x$: Load the value of the input wire l_x to the memory wire M_x .
- $\text{Add}(M_x, M_y) \rightarrow M_z$: Add the values of the memory wires M_x and M_y , and assign the result to the memory wire M_z .
- $\text{Add}(l_x, l_y) \rightarrow l_z$: Add the values of the input wires l_x and l_y , and assign the result to the input wire l_z .
- $\text{Mul}(l_x, M_y) \rightarrow M_z$: Multiply the values of the input wire l_x and memory wire M_y , and assign the result to the memory wire M_z .
- $\text{Output}(M_x, n_{\text{out}}) \rightarrow x \bmod n_{\text{out}}$: Output the value of the memory wire M_x , reduced modulo n_{out} , for some $n_{\text{out}} \leq B_{\text{msg}}$.

The input consists of one integer for each input wire. Whenever the absolute value of any wire exceeds the bound B_{msg} , the output of the execution is $\perp$.

8.4 The Template

As usual in the HSS literature [\[BGI16a, BKS19, OSY21, RS21, ADOS22\]](#), our HSS template supports the class of polynomial-size RMS programs with bound $B_{\text{msg}} \in \mathbb{N}$. Let $\mathcal{S}$ be an LHE scheme with distributed decryption (cf. [Definition 8.1](#)) and large-enough plaintext space $\mathbb{Z}_t$, namely, $t \geq B_{\text{msg}} 2^\kappa$ for statistical security parameter κ . We denote homomorphic addition by $\text{ct}_1 \oplus \text{ct}_2$ and homomorphic multiplication with a plaintext value x by $x \odot \text{ct}$. We also require a rerandomization algorithm $\mathcal{S}.\text{Rerand}(\text{pk}, \text{ct})$ for rerandomizing a ciphertext ct . In group-based cryptosystems (such as the ones we present in this work), $\mathcal{S}.\text{Rerand}$ can typically be implemented by adding a fresh encryption of zero, i.e., $\mathcal{S}.\text{Rerand}(\text{pk}, \text{ct}) := \text{ct} \oplus \mathcal{S}.\text{Enc}(\text{pk}, 0)$.

Ideally, the entries of the secret key should be significantly smaller than the plaintext modulus t , more precisely, $B_{\text{sk}} \leq \frac{t}{B_{\text{msg}} 2^\kappa}$ where B_{sk} is an upper bound on $\|\text{sk}\|_\infty$. When using a cryptosystem where B_{sk} is too large (e.g., our basic cryptosystem), we need to decompose the key into ℓ_d smaller parts for some $\ell_d \geq m$. That is, let $\text{Decomp}, \text{Compose}$ be deterministic polynomial-time algorithms such that:

- $\text{Decomp}(\text{sk})$ outputs a decomposition $(d^{(0)}, \dots, d^{(\ell_d-1)}) \in \mathbb{Z}^{\ell_d}$ with $|d^{(i)}| \leq B_{\text{dig}}$ for all i , where $B_{\text{dig}} \leq \frac{t}{B_{\text{msg}} 2^\kappa}$.
- $\text{Compose}(d^{(0)}, \dots, d^{(\ell_d-1)})$ is a linear function such that, for all secret keys sk , it holds that $\text{Compose}(\text{Decomp}(\text{sk})) = \text{sk}$.

When the secret key's entries are already sufficiently small (e.g., with our ElGamal-style or BHHO-style cryptosystem using suitable parameters), one can use $\ell_d = m$, $\text{Decomp}(\text{sk}) = \text{sk}$, $\text{Compose}(\text{sk}) = \text{sk}$, and $B_{\text{dig}} = B_{\text{sk}}$. Otherwise, Decomp should implement a digit decomposition with base B_{dig} and Compose its inverse. In that case, we choose ℓ_d as small as the bound on B_{dig} permits and then spread the key evenly over the digits, i.e., $B_{\text{dig}} \approx B_{\text{sk}}^{1/\ell_d}$.

With this, our HSS template is defined as follows.

Construction 8.3: Generic HSS Template

Setup(1^λ):

1. Sample $(\text{pk}_S, \text{sk}_S = \mathbf{d}) \leftarrow \mathcal{S}.\text{Gen}(1^\lambda)$.
2. Sample $\llbracket \mathbf{d} \rrbracket_0^{(\mathbb{Z}^m)} \xleftarrow{\$} [B_{\text{sk}} 2^\kappa]^m$ and compute $\llbracket \mathbf{d} \rrbracket_1^{(\mathbb{Z}^m)} := \llbracket \mathbf{d} \rrbracket_0^{(\mathbb{Z}^m)} + \mathbf{d}$ to complete the subtractive sharing.
3. Compute $(d^{(0)}, \dots, d^{(\ell_d-1)}) := \text{Decomp}(\mathbf{d})$.
4. Encrypt $D^{(i)} \leftarrow \mathcal{S}.\text{Enc}(\text{pk}_S, d^{(i)})$ for $i \in [\ell_d]$.
5. Sample a key k_{PRF} for a pseudorandom function (PRF) $\mathcal{F}$ with outputs in the range $[t]$.
6. Let $\text{pk} = (\text{pk}_S, D^{(0)}, \dots, D^{(\ell_d-1)})$ and $\text{ek}_\sigma := (\text{pk}_S, \llbracket \mathbf{d} \rrbracket_\sigma^{(\mathbb{Z}^m)}, k_{\text{PRF}})$ for $\sigma \in \{0, 1\}$.
7. Output $(\text{pk}, (\text{ek}_0, \text{ek}_1))$.

Input(pk, x):

1. Compute $X^{(i)} \leftarrow \mathcal{S}.\text{Rerand}(\text{pk}_S, x \odot D^{(i)})$ for $i \in [\ell_d]$ where $\odot$ denotes homomorphic multiplication by a constant.
2. Output (l_0, l_1) where $\text{l}_0 = \text{l}_1 = (X^{(0)}, \dots, X^{(\ell_d-1)})$.

ConvertInput($\sigma, \text{ek}_\sigma, \text{l} = (X^{(0)}, \dots, X^{(\ell_d-1)})$): Output $\text{Mul}(\sigma, \text{ek}_\sigma, \text{l}, \llbracket \mathbf{d} \rrbracket_\sigma^{(\mathbb{Z}^m)})$.

Add($\sigma, \text{ek}_\sigma, \text{M}_x, \text{M}_y$): Output $\text{M}_z := \text{M}_x + \text{M}_y$ (component-wise addition if $m \geq 2$).

Add($\sigma, \text{ek}_\sigma, \text{l}_x = (X^{(0)}, \dots, X^{(\ell_d-1)}), \text{l}_y = (Y^{(0)}, \dots, Y^{(\ell_d-1)})$):

1. Compute $Z^{(i)} := X^{(i)} \oplus Y^{(i)}$ for $i \in [\ell_d]$.
2. Output $\text{l}_z = (Z^{(0)}, \dots, Z^{(\ell_d-1)})$.

Mul($\sigma, \text{ek}_\sigma, \text{l}_x = (X^{(0)}, \dots, X^{(\ell_d-1)}), \text{M}_y = \llbracket y\mathbf{d} \rrbracket_\sigma^{(\mathbb{Z}^m)}$):

1. Compute $\llbracket zd^{(i)} \rrbracket_\sigma^{(t)} := \mathcal{S}.\text{DDec}(\llbracket y\mathbf{d} \rrbracket_\sigma^{(\mathbb{Z}^m)}, X^{(i)}) + \mathcal{F}_{k_{\text{PRF}}}(\text{id}, i) \bmod t$ for $i \in [\ell_d]$, where id is the unique index of this multiplication.
2. Let $\llbracket zd^{(i)} \rrbracket_\sigma^{(\mathbb{Z})} := \llbracket zd^{(i)} \rrbracket_\sigma^{(t)} \bmod M$, where $M := B_{\text{msg}} B_{\text{dig}} 2^\kappa$, for $i \in [\ell_d]$, i.e., reinterpret the shares over the integers and reduce them modulo M .
3. Compute $\llbracket z\mathbf{d} \rrbracket_\sigma^{(\mathbb{Z}^m)} := \text{Compose}(\llbracket zd^{(0)} \rrbracket_\sigma^{(\mathbb{Z})}, \dots, \llbracket zd^{(\ell_d-1)} \rrbracket_\sigma^{(\mathbb{Z})})$.
4. Output $\text{M}_z := \llbracket z\mathbf{d} \rrbracket_\sigma^{(\mathbb{Z}^m)}$.

Output($\sigma, \text{ek}_\sigma, \text{M}_x = \llbracket x\mathbf{d} \rrbracket_\sigma^{(\mathbb{Z}^m)}$):

1. Encrypt $\text{ct}_1 \leftarrow \mathcal{S}.\text{Enc}^0(\text{pk}_S, 1)$.^a
2. Compute $\llbracket x \rrbracket_\sigma^{(t)} := \mathcal{S}.\text{DDec}(\llbracket x\mathbf{d} \rrbracket_\sigma^{(\mathbb{Z}^m)}, \text{ct}_1)$
3. Output $\llbracket x \rrbracket_\sigma^{(n_{\text{out}})} := \llbracket x \rrbracket_\sigma^{(t)} \bmod n_{\text{out}}$.

^a Here, Enc uses fixed randomness 0, so both parties obtain the same ciphertext ct_1 , which can be reused for all **Output** calls.

The shares constituting a memory value are used as exponents during the next distributed decryption, so their magnitude determines the cost of the exponentiations. The shares that `Mul` obtains from the `DDLog` are of magnitude t each, whereas the values they share are bounded by $B_{\text{msg}}B_{\text{dig}}$, which is why `Mul` reduces them modulo $M = B_{\text{msg}}B_{\text{dig}}2^\kappa$ before composing them. Reducing the shares modulo M preserves the shared value, except with probability $2^{-\kappa+1}$ per digit and multiplication. The reduction matters most where no decomposition is needed and t is dictated by the instantiation rather than by $B_{\text{msg}}B_{\text{sk}}2^\kappa$, as in the Paillier instantiation of our ElGamal-style or BHHO-style cryptosystem, where it shortens the exponents by a factor of eight.

Theorem 8.4. *Let $\mathcal{S}$ be an LHE scheme subject to the above-mentioned conditions and with semantic and circular security, and let $\mathcal{F}$ be a secure PRF. Then [Construction 8.3](#) presents a secure public-key HSS scheme for the class of polynomial-size RMS programs with bound B_{msg} .*

We omit the proof of [Theorem 8.4](#), since it closely follows [\[OSY21, ADOS22\]](#).

HSS Without Circular Security Assumption. As observed in [\[OSY21, ADOS22\]](#), the circular security assumption can be avoided by using a cryptosystem that admits creating encryptions of the secret key without knowing the secret key. Formally, this requires a ppt. algorithm `SkEnc` such that $\text{ct} \leftarrow \text{SkEnc}(\text{pk}, x \in \mathbb{Z}_t, i \in [m])$ always decrypts to x times the i^{th} component of the secret key, i.e., $\text{Dec}(\text{sk}, \text{ct}) = x \cdot d^{(i)} \bmod t$. If such an algorithm exists (which is true in our ElGamal-style and BHHO-style cryptosystems), one can remove $D^{(0)}, \dots, D^{(\ell_d-1)}$ from the public key (hence, no circular security is required) and, in the `Input` subroutine, replace $X^{(i)} \leftarrow \mathcal{S}.\text{Rerand}(\text{pk}_{\mathcal{S}}, x \odot D^{(i)})$ with $X^{(i)} \leftarrow \mathcal{S}.\text{SkEnc}(\text{pk}_{\mathcal{S}}, x, i)$.

9 HSS from Subgroup Indistinguishability

In this section, we define several cryptosystems from the SgI assumption (cf. [Definition 4.4](#)) generically in the subgroup framework. All of our cryptosystems require the discrete logarithm in F to be easy. Our most basic cryptosystem captures, e.g., the Goldwasser–Micali [\[GM82\]](#), the Joye–Libert [\[JL13\]](#), and the Paillier cryptosystem [\[Pai99\]](#). One can choose any one of the cryptosystems presented in this section to instantiate the generic HSS template (cf. [Section 8](#)).

9.1 Basic Cryptosystem

In [Construction 9.2](#), we present what we call our *basic cryptosystem*.

Theorem 9.1. *Let $\mathcal{I}$ be as in [Construction 9.2](#). Then the basic cryptosystem is perfectly correct and semantically secure under the SgI assumption.*

Proof. Using $d \bmod t = 1$ and $d \bmod |H| = 0$, it follows that $\text{ct}^d = (rh^x)^d f^{xd} = f^x$, hence $\text{Dec}(\text{sk}, \text{ct}) = \log_f(\text{ct}^d) = x$. Semantic security follows by a hybrid argument:

- H_0 is the security game with fixed-zero message.
- H_1 is defined like H_0 , except that we replace $y := hf$ with $y := h$. By the SgI assumption, $H_0 \approx_c H_1$. In H_1 , the challenge ciphertext has the form $\text{ct} = rh^{bx}$, which is clearly uniform over H . Hence, the output of H_1 is independent of the challenger’s choice bit.

□

Construction 9.2: Basic Cryptosystem

Let $\mathcal{I}$ be a subgroup framework instantiation with an $\mathcal{I}.\text{TrapdoorGen}$ algorithm, where $\gcd(|H|, t) = 1$.

$\text{Gen}(1^\lambda)$:

1. Sample $(\text{pp} = (G, H, f, t, \hat{N}), |H|) \leftarrow \mathcal{I}.\text{TrapdoorGen}(1^\lambda)$.
2. Sample $h \leftarrow \mathcal{I}.\text{Sample}(H)$ and compute $y := hf$.
3. Compute $d := |H| \cdot (|H|^{-1} \bmod t) \in \mathbb{Z}$.
4. Output $\text{pk} := (\text{pp}, y)$ and $\text{sk} := d$.

$\text{Enc}(\text{pk}, x \in \mathbb{Z}_t)$:

1. Sample $r \leftarrow \mathcal{I}.\text{Sample}(H)$.
2. Output $\text{ct} := ry^x$.

$\text{Dec}(\text{sk}, \text{ct})$:

1. Output $\log_f(\text{ct}^d)$.

On Revealing f . In the basic cryptosystem, the mask h in $y := hf$ serves the purpose of hiding the element f from the adversary. As a consequence, when we remove f from the public key, security even holds under a weaker version of the SgI assumption, where the adversary is not given f . However, in this work we include f in the public key, since the DDLog requires the base f to be public. In this case, the mask h is not required (i.e., y can be hard-coded as $y := f$) and $\mathcal{I}.\text{Sample}$ does not necessarily need to output a uniform distribution (but of course, the SgI assumption, which depends on the distribution of $\mathcal{I}.\text{Sample}$, still needs to hold).

Comparison to [GM82, Ben94, JL13]. When instantiated with $\text{JL}_{t,\ell}$ (cf. Definition 4.11), our basic cryptosystem coincides with Goldwasser–Micali [GM82] when $t = 2$ and with Joye–Libert [JL13] when $t = 2^k$ is a power of two. It can also be seen as a variant of the Benaloh cryptosystem [Ben94] when t is odd. Our cryptosystem is more general than all of these, since first, it works with any plaintext modulus t that contains no large prime factors (so the discrete logarithm in F is easy), and second, it works with any instantiation of the subgroup framework, as long as the SgI assumption holds.

Note that above-mentioned works [GM82, Ben94, JL13] define the secret key and decryption procedure differently than we do: [GM82] (resp. [JL13]) define decryption in terms of computing the Legendre symbol (resp. 2^k -th power residue symbol) of $\text{ct} \bmod p$, where p is one of the prime factors of N . Our decryption procedure from Construction 9.2 is equivalent and generalizes to other instantiations of the subgroup framework. The decryption procedure in [Ben94] computes the discrete logarithm with base f^d instead of f . In our variant, we ensure that $d \bmod t = 1$, so the base during decryption is $f^d = f$. This makes our cryptosystem (when we include f in the public key) compatible with the new DDLog from Section 7, which requires the base to be public.

9.2 ElGamal-style Cryptosystem

In Construction 9.3, we present an *ElGamal-style cryptosystem* based on subgroup indistinguishability. Under the Paillier instantiation, this cryptosystem corresponds to what was called Paillier–ElGamal in [OSY21] (except that our definition supports the small exponent

assumption as an option to reduce the size of the secret key). Compared to the basic cryptosystem, the ElGamal-style cryptosystem has several advantages (which are shared with the DDH-based cryptosystem from [ADOS22]):

1. The same parameters ($\mathbf{pp}$) can be generated once, in a trusted setup, and reused by many parties. In the threshold setting, this enables a distributed key generation in a single round.
2. Using a large-enough plaintext space, the secret key fits into a single ciphertext, eliminating the need for digit decomposition in HSS (or in other applications that require an encryption of the secret key).
3. The circular security assumption can be avoided, as we explain in detail below.

For security, the ElGamal-style cryptosystem requires that H is cyclic and its order contains only superpolynomially large prime factors. This has the effect that a uniformly random element from H is with overwhelming probability a generator of H .

Construction 9.3: ElGamal-style Cryptosystem

Gen(1^λ):

1. Sample $\mathbf{pp} = (G, H, f, t, \hat{N}) \leftarrow \mathcal{I}.\text{Gen}(1^\lambda)$.
2. Sample $h \leftarrow \mathcal{I}.\text{Sample}(H)$.
3. Sample $d \xleftarrow{\$} [\hat{N}(\lambda)]$.
4. Output $\mathbf{pk} := (\mathbf{pp}, h, h^{-d})$ and $\mathbf{sk} := d$.

Enc($\mathbf{pk}, x \in \mathbb{Z}_t$):

1. Sample $r \xleftarrow{\$} [\hat{N}]$.
2. Output $\mathbf{ct} := (h^r, (h^{-d})^r f^x)$.

SkEnc($\mathbf{pk}, x \in \mathbb{Z}_t$):

1. Sample $r \xleftarrow{\$} [\hat{N}]$.
2. Output $\mathbf{ct} := (h^r f^x, (h^{-d})^r)$.

Dec($\mathbf{sk}, \mathbf{ct} = (c_0, c_1)$):

1. Output $\log_f(c_0^d c_1)$.

Theorem 9.4. *Let $\mathcal{I}$ be a subgroup framework instantiation where, always, H is cyclic, $|H|$ contains only superpolynomially large prime factors, and $\gcd(|H|, t) = 1$. Let $\tilde{N} = \tilde{N}(\lambda)$. The ElGamal-style cryptosystem from [Construction 9.3](#) is perfectly correct and semantically secure under the SgI assumption and the small exponent assumption with length $\tilde{N}$.*

Proof. Perfect correctness is easy to see. Semantic security follows by a hybrid argument:

- $\mathbf{H}_0$ is the security game with fixed-zero message.
- $\mathbf{H}_1$ is defined like $\mathbf{H}_0$, except that we replace $d \xleftarrow{\$} [\tilde{N}(\lambda)]$ with $d \xleftarrow{\$} [\hat{N}]$. By the small exponent assumption, $\mathbf{H}_0 \approx_c \mathbf{H}_1$.
- $\mathbf{H}_2$ is defined like $\mathbf{H}_1$, except that we replace h^{-d} with a uniform $h' \xleftarrow{\$} H$ in the public key. h is with overwhelming probability a generator of H , so h^{-d} is close to uniform over H , hence $\mathbf{H}_1 \approx_c \mathbf{H}_2$.
- $\mathbf{H}_3$ is defined like $\mathbf{H}_2$, except that we replace h' with $h'f$ in the public key. By the SgI assumption, $\mathbf{H}_2 \approx_c \mathbf{H}_3$. Now the challenge ciphertext has the form $\mathbf{ct} = (h^r, h'^r f^{r+bx})$. Since the order of H is coprime to t , we can choose the exponent of f independently, which is what we do in the next step:

- H_4 is defined like H_3 , except that we first sample $r' \xleftarrow{\$} [\hat{N}]$ independent of r and replace the challenge ciphertext with $\text{ct} = (h^r, h'^r f^{r'+bx})$. In H_4 , the challenge ciphertext (and hence, the adversary's output) is distributed independent of the challenger's choice bit. $\square$

Encryptions of the Secret Key. Some applications, such as HSS, require working with an encryption of the secret key. Revealing such an encryption is not necessarily secure under semantic security alone, and therefore, commonly requires an additional *circular security* assumption. Our ElGamal-style cryptosystem (similarly to the DDH-based cryptosystem from [ADOS22]) allows someone, who possesses only the public key, to sample a ciphertext $\text{SkEnc}(\text{pk}, x)$ that decrypts to $xd \bmod t$. In many applications (including HSS), $\text{SkEnc}(\text{pk}, x)$ can be used instead of $\text{Enc}(\text{pk}, xd)$, and hence, no circular security assumption is needed.

Semantic security extends to SkEnc in the sense that an adversary who sends x_0, x_1 can neither detect whether x_0 or x_1 was encrypted, nor whether Enc or SkEnc was used by the challenger. In particular, $\text{SkEnc}(\text{pk}, x)$ hides x .

9.3 BHHO-style Cryptosystem

In [BHHO08], Boneh et al. constructed a circular-secure cryptosystem from DDH, which was in [BG10] adapted to use the SgI assumption instead. The proof of circular security relies on the fact that this cryptosystem allows someone, who possesses only the public key, to create an encryption of any of the secret key's *bits*. In **Construction 9.5**, we present a BHHO-style cryptosystem, where the secret key is no longer a bit string, but a string of n digits (bounded by a parameter B). This gives up provable circular security, but reduces the size of the ciphertexts, and still allows one to create encryptions of $(x \text{ times})$ the i^{th} digit of the secret key using the SkEnc procedure. This cryptosystem yields HSS *without circular security assumption and without small exponent assumption*, which was so far only achieved via the cryptosystems from [BHHO08, BG10], which have very large plaintexts.

Construction 9.5: BHHO-style Cryptosystem

$\text{Gen}(1^\lambda)$:

1. Sample $\text{pp} = (G, H, f, t, \hat{N}) \leftarrow \mathcal{I}.\text{Gen}(1^\lambda)$.
2. Sample $h_1, \dots, h_n \leftarrow \mathcal{I}.\text{Sample}(H)$.
3. Sample $d_1, \dots, d_n \xleftarrow{\$} [B]$.
4. Compute $h_0 := \prod_{i=1}^n h_i^{-d_i}$.
5. Output $\text{pk} := (\text{pp}, h_0, h_1, \dots, h_n)$ and $\text{sk} := (d_1, \dots, d_n)$.

$\text{Enc}(\text{pk}, x \in \mathbb{Z}_t)$:

1. Sample $r \xleftarrow{\$} [\hat{N}]$.
2. Output $\text{ct} := (h_0^r f^x, h_1^r, \dots, h_n^r)$.

$\text{SkEnc}(\text{pk}, x \in \mathbb{Z}_t, i \in \{1, \dots, n\})$:

1. Sample $r \xleftarrow{\$} [\hat{N}]$.
2. Output $\text{ct} := (h_0^r, h_1^r, \dots, h_{i-1}^r, h_i^r f^x, h_{i+1}^r, \dots, h_n^r)$.

$\text{Dec}(\text{sk}, \text{ct} = (c_0, c_1, \dots, c_n))$:

1. Output $\log_f(c_0 \prod_{i=1}^n c_i^{d_i})$.

Theorem 9.6. *Let $\mathcal{I}$ be a subgroup framework instantiation where H is always cyclic and $\gcd(|H|, t) = 1$. Let $\text{spf}(|H|)$ denote the smallest prime factor of $|H|$ and $\text{npf}(|H|)$ the number of distinct prime factors. Let B, n be functions in λ , such that*

$$B^n \geq 4^\kappa |H| \quad \text{and} \quad B \cdot \text{npf}(|H|) \cdot (2/\text{spf}(|H|))^n \leq 4^{-\kappa}$$

for all H output by $\mathcal{I}.\text{Gen}(1^\lambda)$ and statistical security parameter κ . Then the BHHO-style cryptosystem from [Construction 9.5](#) is perfectly correct and semantically secure under the Sgl assumption.

Both conditions on B and n refer to $|H|$, which is not public in either of our instantiations. It suffices, however, that whoever picks B and n knows a lower bound on $\text{spf}(|H|)$ as well as upper bounds on $\text{npf}(|H|)$ and $|H|$. The former two are fixed by how the RSA semiprime is sampled: with safe primes p, q in Paillier_ℓ^2 (resp. prime $p'/\tau, q'/\tau$ in $\text{JL}_{t,\ell}^\tau$), $|H|$ is the product of two primes of about $\ell - 1$ (resp. $\ell - \log_2(\tau t)$) bits. The upper bound on $|H|$ is $\hat{N}/t$ in any instantiation.

For the proof of [Theorem 9.6](#), we use the leftover hash lemma [[HILL99](#)] in the following form, which is a special case of [[BDK⁺11](#), Lemma 3.1].

Lemma 9.7 (Leftover Hash Lemma, [[BDK⁺11](#)]). *Let $\mathcal{H}$ be a family of functions from a set $\mathcal{X}$ to a set $\mathcal{Y}$, and let $\rho := \Pr[h(x) = h(x')]$, where $h \xleftarrow{\$} \mathcal{H}$ and $x, x' \xleftarrow{\$} \mathcal{X}$. Then the distribution $(h, h(x))$, where $h \xleftarrow{\$} \mathcal{H}$ and $x \xleftarrow{\$} \mathcal{X}$, is $\frac{1}{2}\sqrt{|\mathcal{Y}|\rho - 1}$ -uniform over $\mathcal{H} \times \mathcal{Y}$.*

Proof of Theorem 9.6. Perfect correctness is again easy to see, so we proceed with semantic security. Consider the hash family

$$\left\{ \begin{array}{l} \mathcal{H} : [B]^n \rightarrow H \\ (d_1, \dots, d_n) \mapsto h_0 = \prod_{i=1}^n h_i^{-d_i} \end{array} \middle| h_1, \dots, h_n \in H \right\},$$

whose collision probability ρ we bound. Let $\mathbf{d} := (d_1, \dots, d_n)$ and $\mathbf{d}' := (d'_1, \dots, d'_n)$ be independent and uniform over $[B]^n$, let $\delta_i := d'_i - d_i$, and let $\gamma := \gcd(\delta_1, \dots, \delta_n, |H|)$. Fixing a generator g of H and writing $h_i = g^{a_i}$ with $a_1, \dots, a_n$ uniform over $\mathbb{Z}_{|H|}$, we have $\mathcal{H}(\mathbf{d}) = \mathcal{H}(\mathbf{d}')$ if and only if $\sum_{i=1}^n a_i \delta_i \equiv 0 \pmod{|H|}$. For fixed $\delta_1, \dots, \delta_n$, the map $(a_1, \dots, a_n) \mapsto \sum_{i=1}^n a_i \delta_i$ is a homomorphism from $\mathbb{Z}_{|H|}^n$ onto the subgroup of $\mathbb{Z}_{|H|}$ generated by γ , which has $|H|/\gamma$ elements. Hence, $|H|\rho$ is the expected value of γ over the choice of $\mathbf{d}$ and $\mathbf{d}'$. We upper-bound this expectation by considering three cases:

- If all δ_i are zero, then $\gamma = |H|$, contributing $|H|/B^n \leq 4^{-\kappa}$ to that expectation.
- If no prime factor of $|H|$ divides all δ_i , then $\gamma = 1$, contributing at most 1.
- Otherwise, some prime factor p of $|H|$ divides every δ_i , while $\delta_j \neq 0$ for some j , hence $p \leq \gamma \leq |\delta_j| < B$. Such a prime p divides every δ_i with probability at most $(\lceil B/p \rceil / B)^n < (2/p)^n \leq (2/\text{spf}(|H|))^n$. Summing over the $\text{npf}(|H|)$ distinct prime factors of $|H|$, this case contributes at most $B \cdot \text{npf}(|H|) \cdot (2/\text{spf}(|H|))^n \leq 4^{-\kappa}$.

Altogether, $|H|\rho \leq 1 + 2 \cdot 4^{-\kappa}$, so the public key $(h_0, h_1, \dots, h_n)$ is $\frac{1}{2}\sqrt{|H|\rho - 1} \leq \frac{1}{2}\sqrt{2 \cdot 4^{-\kappa}} \leq 2^{-\kappa}$ -uniform over H^{n+1} by [Lemma 9.7](#). Now consider the following sequence of hybrids:

- H_0 is the security game with fixed-zero message.
- H_1 is defined like H_0 , except that we replace $h_0 := \prod_{i=1}^n h_i^{-d_i}$ with $h_0 \xleftarrow{\$} H$. By the above observation, $H_0 \approx_c H_1$. The remaining proof is analogous to the proof of [Theorem 9.4](#), so we omit the detailed arguments:
- H_2 is defined like H_1 , except that we replace h_0 with $h_0 f$ in the public key.
- H_3 is defined like H_2 , except that we replace the first component of the challenge ciphertext with $c_0 = h_0^r f^{r'+bx}$. In H_3 , the adversary's output is distributed independently of the challenger's choice bit.

□

9.4 Comparison of HSS Instantiations

In [Table 9.8](#), we compare the resulting HSS instantiations based on our three cryptosystems, using $B_{\text{msg}} = 1$ and $\kappa = 128$. In each case, we consider the $\text{Paillier}_\ell^\tau$ and $\text{JL}_{t,\ell}$ instantiations of the subgroup framework, using an RSA semiprime of length $2\ell = 3072$ and, in the Joye–Libert case, plaintext modulus $t = 2^{384}$. In the Paillier instantiation, the ElGamal-style and BHHO-style cryptosystems require H to be cyclic and therefore use $\tau = 2$, whereas the basic cryptosystem requires only $\gcd(|H|, t) = 1$ and uses $\tau = 1$. For the ElGamal-style cryptosystem, we assume a 256-bit exponent, i.e., $B_{\text{sk}} = 2^{256}$. This satisfies $B_{\text{sk}} \leq \frac{t}{B_{\text{msg}} 2^\kappa}$, as required by the generic HSS template (cf. [Section 8](#)). For the BHHO-style cryptosystem to be secure (without relying on a small exponent assumption), we additionally require $B_{\text{sk}}^n \geq |H| 4^\kappa$, where $|H| \leq \hat{N}/t$. In order to minimize the ciphertext size and the number of ciphertexts required for HSS inputs, we use the smallest possible value of n where both inequalities hold. (The resulting parameter sets also satisfy the second inequality required by [Theorem 9.6](#).)

Our comparison shows that the ElGamal-style cryptosystem yields the best HSS performance. This efficiency stems from the use of the small exponent assumption, which significantly speeds up decryption, while requiring only two ciphertexts per HSS input share.

At our parameters, one group operation takes $0.56 \mu\text{s}$ in $\mathbb{Z}_N^*$ versus $1.99 \mu\text{s}$ in $\mathbb{Z}_{N^2}^*$, so the Joye–Libert instantiation enjoys $3.5\times$ faster group operations, but in return requires a more expensive DDLog. The net effect is that ElGamal-style HSS from Joye–Libert is around $1.8\times$ slower than from Paillier. These Joye–Libert measurements are based on our new DDLog algorithm, which improves performance by a measured $16\times$ over [\[ADOS22\]](#). Although Joye–Libert is still slower than Paillier, it yields smaller HSS input shares (using the ElGamal-style cryptosystem), making it a competitive choice for applications where minimizing the share size is more important than optimizing runtime. Thanks to our new DDLog, this consideration will more often lean toward reducing the share size.

Table 9.8. Efficiency comparison of our HSS instantiations with $B_{\text{msg}} = 1$, $\kappa = 128$, $2\ell = 3072$, and (in the JL case) $t = 2^{384}$. Each runtime is the work of one party and the mean of six evaluations, measured single-threaded on a standard laptop (Intel Core Ultra 7 258V). All algorithms have been implemented in constant time, i.e., run in time independent of the secret key shares.

Cryptosystem	Instantiation	$\log_2 B_{\text{sk}}$	Ciphertext size		HSS Input share size		DDec runtime (ms)			HSS Mul runtime (ms) [§]
			# GE [#]	kbit	# ct [*]	kbit	Exp [†]	DDLog [‡]	$\sum$	
Basic (Construction 9.2)	Paillier_ℓ^1	6 144	1	6	3	18	15.55	0.24	15.80	47.40
	$\text{JL}_{t,\ell}^1$	2 688	1	3	11	33	2.08	2.04	4.13	45.38
ElGamal-style (Construction 9.3)	Paillier_ℓ^2	256	2	12	2	24	1.09	0.24	1.33	2.67
	$\text{JL}_{t,\ell}^1$	256	2	6	2	12	0.34	2.03	2.37	4.73
BHHO-style (Construction 9.5)	Paillier_ℓ^2	1 663	3	18	3	54	5.68	0.24	5.92	17.78
	$\text{JL}_{t,\ell}^1$	256	11	33	11	363	1.04	2.03	3.07	33.80

[#] Number of group elements per ciphertext.

^{*} Number of ciphertexts per HSS input share.

[†] Time for all exponentiations during distributed decryption.

[‡] Time for all DDLog calls during distributed decryption. The DDLog algorithm for Paillier groups [\[OSY21\]](#) needs only one division over $\mathbb{Z}_N$, which is a modular inversion and no exponentiation at all; that is why it stays far below the Joye–Libert figure without being free.

[§] Time per multiplication gate, which is #ct distributed decryptions and therefore #ct · $\sum$ up to the cost of the PRF and the share bookkeeping.

10 PCF Definitions

In this section, we present the PCF definitions underlying the constructions in [Section 11](#).

Informally, a PCF [\[BCG⁺20a\]](#) consists of a setup algorithm **Setup** that publishes public parameters $\mathbf{pp}$, a key generation algorithm **Gen** that outputs a key pair k_0, k_1 , and an evaluation algorithm **Eval**. For a sequence of distinct inputs $x^{(i)}$, the outputs $(y_0^{(i)}, y_1^{(i)}) := (\text{Eval}_{k_0}(x^{(i)}), \text{Eval}_{k_1}(x^{(i)}))$ must be indistinguishable from independent samples of a *target correlation* $\mathcal{Y}(\mathbf{pp}, \mathbf{mk})$, even given the inputs $x^{(i)}$. This indistinguishability should hold not only for external observers, but also for “insiders” who possess one of the two keys. Of course, a party P_σ (for $\sigma \in \{0, 1\}$) holding k_σ can compute $y_\sigma^{(i)} := \text{Eval}_{k_\sigma}(x^{(i)})$ and thus distinguish it from a random sample. Therefore, the security definition instead demands that P_σ cannot distinguish the counterparty’s actual outputs $y_{1-\sigma}^{(i)}$ from ideal-model outputs that are reverse-sampled based on $y_\sigma^{(i)}$. For this to be well-defined, the target correlation of a PCF must be reverse-sampleable:

Definition 10.1 (Reverse-Sampleable Correlation). *Let $\mathcal{P} = \mathcal{P}(\lambda)$ be a set of admissible public parameters, let $\mathcal{M}(\mathbf{pp}) = \mathcal{M}_0(\mathbf{pp}) \times \mathcal{M}_1(\mathbf{pp})$ be a set of allowed master keys for each $\mathbf{pp} \in \mathcal{P}$, and let $\mathcal{Y}$ be a ppt. algorithm that, on input $\mathbf{pp} \in \mathcal{P}$ and $\mathbf{mk} \in \mathcal{M}(\mathbf{pp})$, outputs a pair (y_0, y_1) . The tuple $(\mathcal{P}, \mathcal{M}, \mathcal{Y})$ is a reverse-sampleable correlation if there exist a ppt. algorithm **RSample** and a negligible function ν such that, for all λ , all $\mathbf{pp} \in \mathcal{P}$, all $\mathbf{mk} \in \mathcal{M}(\mathbf{pp})$, and all $\sigma \in \{0, 1\}$ (corrupted party’s index), the statistical distance between the distributions*

$$(y_\sigma, y_{1-\sigma}) \quad \text{and} \quad (y_\sigma, y^*)$$

is at most $\nu(\lambda)$, where $(y_0, y_1) \leftarrow \mathcal{Y}(\mathbf{pp}, \mathbf{mk})$ and $y^* \leftarrow \text{RSample}(\mathbf{pp}, \mathbf{mk}, \sigma, y_\sigma)$. We require each set $\mathcal{M}_\sigma(\mathbf{pp})$ to contain a canonical master-key share $\widehat{\mathbf{mk}}_\sigma$, computable in polynomial time from $\mathbf{pp}$ and σ .

All correlations mentioned in this work are reverse-sampleable. More generally, all *additive correlations* – i.e., correlations that output additive secret shares of correlated values – are reverse-sampleable [\[BCG⁺19b\]](#).

The public parameters are generated by the **Setup** algorithm of the PCF ([Definition 10.2](#)), and the correlation only says which ones it admits. A correlation that needs no public parameters therefore sets $\mathcal{P} = \{0, 1\}^*$ and lets $\mathcal{Y}$ ignore its $\mathbf{pp}$ argument. For correlations that do not require a master key, we usually avoid explicitly specifying $\mathcal{M} = \{(\perp, \perp)\}$, and we omit the $\mathbf{mk}$ argument from all algorithms.

Our PCF security definition requires (in simplified notation) that

$$\left(k_\sigma, (x^{(i)}, y_{1-\sigma}^{(i)})_i\right) \approx_c \left(k'_\sigma, (x^{(i)}, \text{RSample}(\mathbf{pp}, \mathbf{mk}, \sigma, y_\sigma^{(i)}))_i\right),$$

where $(k_0, k_1) \leftarrow \text{Gen}(\mathbf{pp}, \text{td}, \mathbf{mk})$ and k'_σ is a key generated from the canonical share $\widehat{\mathbf{mk}}_{1-\sigma}$ in place of the counterparty’s actual one. This captures two guarantees at once: first, even given access to several input–output pairs $(x^{(i)}, y_{1-\sigma}^{(i)})$, the party P_σ cannot distinguish between the real model where the outputs are from the PCF and the ideal model where they are random reverse samples. Second, the key k_σ cannot leak anything about the master-key share $\mathbf{mk}_{1-\sigma}$ held by the counterparty. We give the formal PCF definitions below.

10.1 Weak and Strong PCFs

We distinguish between weak and strong PCFs. A weak PCF requires the inputs $x^{(i)}$ to be distributed uniformly at random over some input domain $\mathcal{X}$ to provide security guarantees.

In contrast, a strong PCF permits adversarially chosen inputs. Note that a weak PCF can be upgraded to a strong PCF in the random oracle model (ROM) by applying the random oracle to the inputs [BCG⁺20a].

Definition 10.2 (Weak Pseudorandom Correlation Function). *A PCF with input domain $\mathcal{X} = \mathcal{X}(\lambda)$ for a reverse-sampleable correlation $(\mathcal{P}, \mathcal{M}, \mathcal{Y})$ consists of three algorithms (Setup, Gen, Eval) such that:*

- *Setup(1^λ) is a ppt. setup algorithm that outputs public parameters $\mathbf{pp} \in \mathcal{P}$ together with a trapdoor $\mathbf{td}$.*
- *Gen($\mathbf{pp}, \mathbf{td}, \mathbf{mk}$) is a ppt. key generation algorithm that outputs a key pair k_0, k_1 .*
- *Eval($\mathbf{pp}, \sigma, k_\sigma, \cdot$) is a polynomial-time deterministic algorithm that takes an input from $\mathcal{X}$. We also use the abbreviated notation $\text{Eval}_{k_\sigma}(\cdot) := \text{Eval}(\mathbf{pp}, \sigma, k_\sigma, \cdot)$.*

The trapdoor $\mathbf{td}$ is passed to Gen but withheld from the adversary. We say that the PCF has a transparent setup if $\mathbf{td} = \perp$, in which case $\mathbf{pp}$ is a common reference string (CRS). The triple (Setup, Gen, Eval) is an (N, B, ε) -secure weak PCF if the following two properties hold:

- **Pseudorandom $\mathcal{Y}$ -Correlated Outputs.** *For every non-uniform probabilistic adversary $\mathcal{A}$ with runtime bound B , it holds for sufficiently large λ that*

$$|\Pr[\text{Game}_{\mathcal{A},0}^{\text{pr}}(\lambda) = 1] - \Pr[\text{Game}_{\mathcal{A},1}^{\text{pr}}(\lambda) = 1]| \leq \varepsilon(\lambda)$$

with $\text{Game}_{\mathcal{A},0}^{\text{pr}}, \text{Game}_{\mathcal{A},1}^{\text{pr}}$ defined as follows, where $\mathcal{A}$ is stateful across its invocations and its first output is a master key $\mathbf{mk} \in \mathcal{M}(\mathbf{pp})$:

<u>$\text{Game}_{\mathcal{A},0}^{\text{pr}}(\lambda)$:</u>	<u>$\text{Game}_{\mathcal{A},1}^{\text{pr}}(\lambda)$:</u>
$(\mathbf{pp}, \mathbf{td}) \leftarrow \text{Setup}(1^\lambda)$	$(\mathbf{pp}, \mathbf{td}) \leftarrow \text{Setup}(1^\lambda)$
For $i \in [N(\lambda)]$: $x^{(i)} \xleftarrow{\$} \mathcal{X}(\lambda)$	For $i \in [N(\lambda)]$: $x^{(i)} \xleftarrow{\$} \mathcal{X}(\lambda)$
$\mathbf{mk} \leftarrow \mathcal{A}(1^\lambda, \mathbf{pp}, (x^{(i)})_{i \in [N(\lambda)]})$	$\mathbf{mk} \leftarrow \mathcal{A}(1^\lambda, \mathbf{pp}, (x^{(i)})_{i \in [N(\lambda)]})$
For $i \in [N(\lambda)]$:	$(k_0, k_1) \leftarrow \text{Gen}(\mathbf{pp}, \mathbf{td}, \mathbf{mk})$
$(y_0^{(i)}, y_1^{(i)}) \leftarrow \mathcal{Y}(\mathbf{pp}, \mathbf{mk})$	For $i \in [N(\lambda)]$:
Return $b' \leftarrow \mathcal{A}((y_0^{(i)}, y_1^{(i)})_{i \in [N(\lambda)]})$	$y_0^{(i)} := \text{Eval}_{k_0}(x^{(i)})$
	$y_1^{(i)} := \text{Eval}_{k_1}(x^{(i)})$
	Return $b' \leftarrow \mathcal{A}((y_0^{(i)}, y_1^{(i)})_{i \in [N(\lambda)]})$

- **Security.** *For all $\sigma \in \{0, 1\}$ and every non-uniform probabilistic adversary $\mathcal{A}$ with runtime bound B , it holds for sufficiently large λ that*

$$|\Pr[\text{Game}_{\mathcal{A},\sigma,0}^{\text{sec}}(\lambda) = 1] - \Pr[\text{Game}_{\mathcal{A},\sigma,1}^{\text{sec}}(\lambda) = 1]| \leq \varepsilon(\lambda)$$

with $\text{Game}_{\mathcal{A},\sigma,0}^{\text{sec}}, \text{Game}_{\mathcal{A},\sigma,1}^{\text{sec}}$ defined as follows, where $\mathcal{A}$ is stateful across its invocations and its first output is a master key $\mathbf{mk} \in \mathcal{M}(\mathbf{pp})$:

<u>$\text{Game}_{\mathcal{A},\sigma,0}^{\text{sec}}(\lambda)$:</u>	<u>$\text{Game}_{\mathcal{A},\sigma,1}^{\text{sec}}(\lambda)$:</u>
$(\mathbf{pp}, \mathbf{td}) \leftarrow \text{Setup}(1^\lambda)$	$(\mathbf{pp}, \mathbf{td}) \leftarrow \text{Setup}(1^\lambda)$
For $i \in [N(\lambda)]$: $x^{(i)} \xleftarrow{\$} \mathcal{X}(\lambda)$	For $i \in [N(\lambda)]$: $x^{(i)} \xleftarrow{\$} \mathcal{X}(\lambda)$
$\mathbf{mk} \leftarrow \mathcal{A}(1^\lambda, \mathbf{pp}, (x^{(i)})_{i \in [N(\lambda)]})$	$\mathbf{mk} \leftarrow \mathcal{A}(1^\lambda, \mathbf{pp}, (x^{(i)})_{i \in [N(\lambda)]})$
$\mathbf{mk}'_\sigma := \mathbf{mk}_\sigma; \mathbf{mk}'_{1-\sigma} := \widehat{\mathbf{mk}}_{1-\sigma}$	$\mathbf{mk}'_\sigma := \mathbf{mk}_\sigma; \mathbf{mk}'_{1-\sigma} := \mathbf{mk}_{1-\sigma}$
$(k_0, k_1) \leftarrow \text{Gen}(\mathbf{pp}, \mathbf{td}, (\mathbf{mk}'_0, \mathbf{mk}'_1))$	$(k_0, k_1) \leftarrow \text{Gen}(\mathbf{pp}, \mathbf{td}, (\mathbf{mk}'_0, \mathbf{mk}'_1))$
For $i \in [N(\lambda)]$:	For $i \in [N(\lambda)]$:
$y_\sigma^{(i)} := \text{Eval}_{k_\sigma}(x^{(i)})$	$y_{1-\sigma}^{(i)} := \text{Eval}_{k_{1-\sigma}}(x^{(i)})$
$y_{1-\sigma}^{(i)} \leftarrow \text{RSample}(\mathbf{pp}, \mathbf{mk}, \sigma, y_\sigma^{(i)})$	Return $b' \leftarrow \mathcal{A}(k_\sigma, (y_{1-\sigma}^{(i)})_{i \in [N(\lambda)]})$
Return $b' \leftarrow \mathcal{A}(k_\sigma, (y_{1-\sigma}^{(i)})_{i \in [N(\lambda)]})$	

Definition 10.3 (Strong Pseudorandom Correlation Function). Let $\mathcal{X}$, $(\mathcal{P}, \mathcal{M}, \mathcal{Y})$, and $(\text{Setup}, \text{Gen}, \text{Eval})$ be as in [Definition 10.2](#). The triple $(\text{Setup}, \text{Gen}, \text{Eval})$ is an (N, B, ε) -secure strong PCF if the following two properties hold:

- **Strong Pseudorandom $\mathcal{Y}$ -Correlated Outputs.** For every non-uniform probabilistic adversary $\mathcal{A}$ with runtime bound B that makes at most $N(\lambda)$ oracle queries, it holds for sufficiently large λ that

$$|\Pr[\text{Game}_{\mathcal{A},0}^{\text{pr}}(\lambda) = 1] - \Pr[\text{Game}_{\mathcal{A},1}^{\text{pr}}(\lambda) = 1]| \leq \varepsilon(\lambda)$$

in the following security game, where $\mathcal{A}$ is stateful across its invocations and its first output is a master key $\text{mk} \in \mathcal{M}(\text{pp})$:

$\text{Game}_{\mathcal{A},b}^{\text{pr}}(\lambda)$:	$O_0^{\text{pr}}(x \in \mathcal{X}(\lambda))$:	$O_1^{\text{pr}}(x \in \mathcal{X}(\lambda))$:
$(\text{pp}, \text{td}) \leftarrow \text{Setup}(1^\lambda)$	If $(x, y_0, y_1) \in Q$:	For $\sigma = 0, 1$:
$\text{mk} \leftarrow \mathcal{A}(1^\lambda, \text{pp})$	Return (y_0, y_1)	$y_\sigma := \text{Eval}_{k_\sigma}(x)$
If $b = 0$:	Else:	Return (y_0, y_1)
$Q := \emptyset$	$(y_0, y_1) \leftarrow \mathcal{Y}(\text{pp}, \text{mk})$	
Else:	$Q := Q \cup \{(x, y_0, y_1)\}$	
$(k_0, k_1) \leftarrow \text{Gen}(\text{pp}, \text{td}, \text{mk})$	Return (y_0, y_1)	
Return $b' \leftarrow \mathcal{A}_b^{O_b^{\text{pr}}}$		

- **Strong Security.** For all $\sigma \in \{0, 1\}$ and every non-uniform probabilistic adversary $\mathcal{A}$ with runtime bound B that makes at most $N(\lambda)$ oracle queries, it holds for sufficiently large λ that

$$|\Pr[\text{Game}_{\mathcal{A},\sigma,0}^{\text{sec}}(\lambda) = 1] - \Pr[\text{Game}_{\mathcal{A},\sigma,1}^{\text{sec}}(\lambda) = 1]| \leq \varepsilon(\lambda)$$

in the following security game, where $\mathcal{A}$ is stateful across its invocations and its first output is a master key $\text{mk} \in \mathcal{M}(\text{pp})$:

$\text{Game}_{\mathcal{A},\sigma,b}^{\text{sec}}(\lambda)$:	$O_{\sigma,0}^{\text{sec}}(x \in \mathcal{X}(\lambda))$:	$O_{\sigma,1}^{\text{sec}}(x \in \mathcal{X}(\lambda))$:
$(\text{pp}, \text{td}) \leftarrow \text{Setup}(1^\lambda)$	If $(x, y_{1-\sigma}) \in Q$:	$y_{1-\sigma} := \text{Eval}_{k_{1-\sigma}}(x)$
$\text{mk} \leftarrow \mathcal{A}(1^\lambda, \text{pp})$	Return $y_{1-\sigma}$	Return $y_{1-\sigma}$
$\text{mk}'_\sigma := \text{mk}_\sigma$	Else:	
If $b = 0$:	$y_\sigma := \text{Eval}_{k_\sigma}(x)$	
$\text{mk}'_{1-\sigma} := \widehat{\text{mk}}_{1-\sigma}; Q := \emptyset$	$y_{1-\sigma} \leftarrow \text{RSample}(\text{pp}, \text{mk}, \sigma, y_\sigma)$	
Else:	$Q := Q \cup \{(x, y_{1-\sigma})\}$	
$\text{mk}'_{1-\sigma} := \text{mk}_{1-\sigma}$	Return $y_{1-\sigma}$	
$(k_0, k_1) \leftarrow \text{Gen}(\text{pp}, \text{td}, (\text{mk}'_0, \text{mk}'_1))$		
Return $b' \leftarrow \mathcal{A}^{O_{\sigma,b}^{\text{sec}}}(k_\sigma)$		

Historical Notes. PCFs were originally introduced in [\[BCG⁺20b\]](#). However, as pointed out in [\[KOR23\]](#), the original framework contains a flaw that makes it impossible to instantiate with correlations that require a master key, such as authenticated Beaver triples and VOLE. Furthermore, even without this issue, the corresponding (weak and strong) PCF definitions in [\[BCG⁺20b\]](#) assumed mk to be generated as part of the PCF key generation (Gen), making it impossible to “program” an externally generated master key.

Both of these issues were fixed in [\[KOR23\]](#) through a revised definition where the master key is “programmable”, i.e., mk is allowed to have an arbitrary distribution and is provided as an input to both Setup and Gen . This permits, for example, the reuse of the same VOLE secret $\text{mk} = (\perp, \Delta)$ across multiple PCF instances. In [\[KOR23\]](#), the Setup algorithm takes

on a new role: instead of sampling the master key (as in [BCG⁺20b]), it serves the purpose of generating parameters that cannot be programmed, because they must follow a specific distribution. For instance, the PCF for VOLE from [OSY21] produces outputs over $\mathbb{Z}_N$ where the parameter N is an RSA semiprime, and where security depends crucially on the fact that one of the two parties does not know the factorization of N .

Our definitions above follow the spirit of [KOR23] but differ in three key aspects. First, the security definition in [KOR23] requires the existence of a simulator to sample the simulated key, which is conceptually more involved, as it demands constructing a suitable simulator when proving the security of a PCF. In contrast, our definitions explicitly specify how the simulated key is generated: by running **Gen** on the canonical master-key share $\widehat{\mathbf{mk}}_{1-\sigma}$ in place of the counterparty's actual one. Second, **Setup** belongs to the PCF rather than to the correlation, and the correlation only fixes the set $\mathcal{P}$ of public parameters it admits. This decouples the two: a PCF may use a CRS even for a correlation that specifies none, and the parameters no longer have to be named before the master key, which is what lets the adversary choose the master key adaptively. It also removes the need for a correlation to prescribe a *distribution* over parameters, which would unnecessarily rule out PCFs that sample them differently. Third, the two properties are quantified over all master keys in [KOR23], whereas ours let the adversary choose the master key after it has seen $\mathbf{pp}$ and the inputs, as discussed above.

11 PCFs from Subgroup Indistinguishability

This section is structured as follows: [Section 11.1](#) presents our PCF for (module) VOLE. [Section 11.2](#) modifies the OT construction of prior works so that its security can be proven, yielding a PCF for 1-out-of- t OT. [Section 11.3](#) explains how to implement a public-key setup for our PCFs. [Section 11.4](#) provides an extension to a (non-public-key) PCF for OLE and degree-two correlations. Finally, [Section 11.5](#) reports the key sizes and the measured runtimes of our PCFs.

11.1 PCF for (Module) VOLE

Definition 11.1 ((Module) VOLE Correlation). *Let $n \in \mathbb{N}_{\geq 1}$. The module VOLE correlation $n\text{-mVOLE} = (\mathcal{P}, \mathcal{M}, \mathcal{Y})$ is defined as follows:*

- $\mathcal{P}$ is the set of public parameters that specify a modulus $t \in \mathbb{N}_{\geq 2}$, which we denote by $t(\mathbf{pp})$.
- $\mathcal{M}(\mathbf{pp}) = \{\perp\} \times \mathbb{Z}_t^n$ (set of allowed master keys) for $t = t(\mathbf{pp})$, with canonical master-key shares $\widehat{\mathbf{mk}}_0 = \perp$ and $\widehat{\mathbf{mk}}_1 = \mathbf{0}$.
- $\mathcal{Y}(\mathbf{pp}, \mathbf{mk})$: Let $t := t(\mathbf{pp})$, parse $\mathbf{mk} =: (\perp, \Delta)$, sample $a \xleftarrow{\$} \mathbb{Z}_t$, $\mathbf{z}_0 \xleftarrow{\$} \mathbb{Z}_t^n$, compute $\mathbf{z}_1 := \mathbf{z}_0 + a\Delta \bmod t \in \mathbb{Z}_t^n$, and output $((a, \mathbf{z}_0), (\Delta, \mathbf{z}_1))$.

We write $(t, n)\text{-mVOLE}$ for the restriction of $n\text{-mVOLE}$ to those $\mathbf{pp} \in \mathcal{P}$ with $t(\mathbf{pp}) = t$.

The module VOLE correlation is reverse-sampleable via the algorithm **RSample** that proceeds, on input $(\mathbf{pp}, \mathbf{mk} = (\perp, \Delta), \sigma, y_\sigma)$ with $t = t(\mathbf{pp})$, as follows: If $\sigma = 0$, parse $y_\sigma =: (a, \mathbf{z}_0) \in \mathbb{Z}_t \times \mathbb{Z}_t^n$ and output $(\Delta, a\Delta + \mathbf{z}_0)$; if $\sigma = 1$, parse $y_\sigma =: (\Delta, \mathbf{z}_1) \in (\mathbb{Z}_t^n)^2$, sample $a \xleftarrow{\$} \mathbb{Z}_t$, and output $(a, \mathbf{z}_1 - a\Delta)$.

In [Construction 11.2](#), we present a PCF for $n\text{-mVOLE}$ for any $n \in \mathbb{N}_{\geq 1}$ from any subgroup framework instantiation, where the modulus t is the order of $F = \langle f \rangle$. Our construction is a simple generalization of the PCFs from [OSY21]. Namely, when using the Paillier instantiation and $n = 1$, we obtain the PCF for $(N, 1)\text{-mVOLE}$ from [OSY21]. When instead using the Joye–Libert instantiation, $t = 2$, and $n = \lambda$, then the basic cryptosystem coincides with the Goldwasser–Micali cryptosystem [GM82] and we obtain the PCF for $(2, \lambda)\text{-mVOLE}$ from [OSY21]. The latter was presented as a PCF for XOR-correlated OT in their work, but XOR-correlated OT is equivalent to $(2, \lambda)\text{-mVOLE}$. Our PCF is more general, as it works with any

subgroup framework instantiation that satisfies the PC-SgI assumption and enjoys an efficient DDLog. In particular, one can employ our PCF with $t \mid 6983776800 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$ under the JL instantiation. Other values of t are currently not supported (but might be supported in the future) due to the lack of an efficient algorithm for computing the t^{th} power residue symbol, required for public-coin sampling from G , as discussed in [Section 5](#).

Construction 11.2: PCF for mVOLE

Parameters:

- $n = n(\lambda) \in \mathbb{N}_{\geq 1}$.
- $\mathcal{I}$ is a subgroup framework instantiation with $\mathcal{I}.\text{TrapdoorGen}$, $\mathcal{I}.\text{SampleG}$, and $\mathcal{I}.\text{DDLog}$ algorithms, where $\gcd(|H|, t) = 1$.
- $\ell_r \in \text{poly}(\lambda)$ such that $\mathcal{I}.\text{SampleG}$ uses at most ℓ_r bits of randomness.
- WPRF is a WPRF with input space $\{0, 1\}^{\leq \ell_r}$ and range $\mathbb{Z}_t^n$.

PCF^{mVOLE}.Setup(1^λ):

Output $(\text{pp} = (G, H, f, t, \hat{N}), |H|) \leftarrow \mathcal{I}.\text{TrapdoorGen}(1^\lambda)$.

PCF^{mVOLE}.Gen($\text{pp}, |H|, (\perp, \Delta)$):

1. Parse $\text{pp} = (G, H, f, t, \hat{N})$.
2. Compute $d := |H| \cdot (|H|^{-1} \bmod t) \in \mathbb{Z}$.
3. Sample $\llbracket \Delta d \rrbracket_0^{(\mathbb{Z}^n)} \xleftarrow{\$} [t\hat{N}2^\kappa]^n$, and compute $\llbracket \Delta d \rrbracket_1^{(\mathbb{Z}^n)} := \llbracket \Delta d \rrbracket_0^{(\mathbb{Z}^n)} + \Delta d$ to complete the subtractive sharing.
4. Sample $k_{\text{WPRF}} \leftarrow \text{WPRF}.\text{Gen}(1^\lambda)$.
5. Output (k_0, k_1) , where $k_0 := (k_{\text{WPRF}}, \llbracket \Delta d \rrbracket_0^{(\mathbb{Z}^n)}, d)$ and $k_1 := (k_{\text{WPRF}}, \llbracket \Delta d \rrbracket_1^{(\mathbb{Z}^n)}, \Delta)$.

PCF^{mVOLE}.Eval($\text{pp}, \sigma, k_\sigma, r \in \{0, 1\}^{\ell_r}$):

1. Compute $g := \mathcal{I}.\text{SampleG}^r(\text{pp})$.
2. For $i = 0, \dots, n-1$: Compute $z_{\sigma, i} := \mathcal{I}.\text{DDLog}(\text{pp}, \sigma, g^{\llbracket \Delta_i d \rrbracket_\sigma^{(\mathbb{Z})}})$.
3. Let $\mathbf{z}_\sigma := (z_{\sigma, 0}, \dots, z_{\sigma, n-1}) + \text{WPRF}.\text{Eval}_{k_{\text{WPRF}}}(r) \bmod t$.
4. If $\sigma = 0$: Compute $a := \log_f(g^d)$ and output $(a, \mathbf{z}_0)$.
If $\sigma = 1$: Output $(\Delta, \mathbf{z}_1)$.

Theorem 11.3. *Let $n, \mathcal{I}, \text{WPRF}, \text{PCF}^{\text{mVOLE}}$ be as in [Construction 11.2](#). Then, under the PC-SgI assumption, $\text{PCF}^{\text{mVOLE}}$ is a weak PCF for n -mVOLE.*

Proof. We will prove the two properties from [Definition 10.2](#) in turn.

Pseudorandom $\mathcal{Y}$ -Correlated Outputs. We prove this property via a sequence of hybrids presented in [Experiment 11.4](#), where the superscript (i) refers to the respective variable from the i^{th} iteration of the for loop.

H_0 corresponds to the real game, i.e., $H_0 \equiv \text{Game}_{\mathcal{A}, 1}^{\text{pr}}$. We have inlined the definition of $\text{PCF}^{\text{mVOLE}}.\text{Eval}$. The variables k_{WPRF} , $\llbracket \Delta_j d \rrbracket_0^{(\mathbb{Z})}$, $\llbracket \Delta_j d \rrbracket_1^{(\mathbb{Z})}$, and d refer to the respective values contained in k_0 and k_1 . It holds that $\llbracket \Delta_j d \rrbracket_1^{(\mathbb{Z})} = \llbracket \Delta_j d \rrbracket_0^{(\mathbb{Z})} + \Delta_j d$, hence $g^{\llbracket \Delta_j d \rrbracket_1^{(\mathbb{Z})}} = g^{\llbracket \Delta_j d \rrbracket_0^{(\mathbb{Z})}} g^{\Delta_j d} = g^{\llbracket \Delta_j d \rrbracket_0^{(\mathbb{Z})}} f^{a \Delta_j}$, and by [Definition 4.2](#) it then follows that $z_{1, j} - z_{0, j} \equiv a \Delta_j \pmod{t}$, hence $\mathbf{z}_0 = \mathbf{z}_1 - a \Delta \bmod t$. Moreover, it holds that $g^d = \pi_F(g)$, as already argued in the correctness proof of the basic cryptosystem. Therefore, $H_0 \equiv H_1$. Now we define H_2 (which we omit from [Experiment 11.4](#) for brevity) like H_1 , except that we sample $\mathbf{z}_1$ uniformly from $\mathbb{Z}_t^n$ (independently in each iteration of the for loop). $H_1 \approx_c H_2$ holds since $\mathcal{F}$ is a WPRF,

whose key k_{WPRF} is sampled independently of Δ . Finally, we define H_3 like H_2 , except that we sample a uniformly from $\mathbb{Z}_t$ (independently in each iteration of the for loop). $H_2 \approx_c H_3$ holds since, under the PC-SgI assumption, $r \mapsto \pi_F(\mathcal{I}.\text{SampleG}^r(\text{pp}))$ is a WPRF by [Theorem 4.9](#), even given pp . The latter hybrid corresponds exactly to the ideal game, i.e., $H_3 \equiv \text{Game}_{\mathcal{A},0}^{\text{pr}}$.

Security. We prove this property from [Definition 10.2](#) separately for $\sigma = 0, 1$.

For $\sigma = 0$, we present a sequence of hybrids in [Experiment 11.5](#). H_0 corresponds to the real game, i.e., $H_0 \equiv \text{Game}_{\mathcal{A},0,1}^{\text{sec}}$. By the same argument as above, it holds that $z_1 = a\Delta + z_0 \bmod t$, hence $H_0 \equiv H_1$. Now we define H_2 (which we omit from [Experiment 11.5](#) for brevity) like H_1 , except that we invoke the PCF key generation with the canonical master-key share $\widehat{\text{mk}}_1 = \mathbf{0}$ in place of Δ . Since only the PCF key k_0 (which statistically hides Δ) is used, it follows that $H_1 \approx_s H_2$. The latter corresponds exactly to the ideal game, i.e., $H_2 \equiv \text{Game}_{\mathcal{A},0,0}^{\text{sec}}$.

For $\sigma = 1$, we present a sequence of hybrids in [Experiment 11.6](#). H_0 again corresponds to the real game, i.e., $H_0 \equiv \text{Game}_{\mathcal{A},1,1}^{\text{sec}}$. By the same arguments as above, it holds that $z_0 = z_1 - a\Delta \bmod t$ and $g^d = \pi_F(g)$, hence $H_0 \equiv H_1$. Now we define H_2 (which we omit from [Experiment 11.6](#) for brevity) like H_1 , except that we sample a uniformly from $\mathbb{Z}_t$ (independently in each iteration of the for loop). $H_1 \approx_c H_2$ holds since, under the PC-SgI assumption, $r \mapsto \pi_F(\mathcal{I}.\text{SampleG}^r(\text{pp}))$ is a WPRF by [Theorem 4.9](#), even given pp . The latter hybrid corresponds exactly to the ideal game, i.e., $H_2 \equiv \text{Game}_{\mathcal{A},1,0}^{\text{sec}}$.

Experiment 11.4: Hybrids for the Pseudorandom $\mathcal{Y}$ -Correlated Outputs of $\text{PCF}^{\text{mVOLE}}$

$H_0(\lambda)$:

```

(pp, |H|) ← PCFmVOLE.Setup(1λ)
For  $i \in [N]$ :  $r^{(i)} \xleftarrow{\$} \{0, 1\}^{\ell_r}$ 
 $(\perp, \Delta) \leftarrow \mathcal{A}(1^\lambda, \text{pp}, (r^{(i)})_{i \in [N]})$ 
 $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, |H|, (\perp, \Delta))$ 
For  $i \in [N]$ :
   $g := \mathcal{I}.\text{SampleG}^{r^{(i)}}(\text{pp})$ 
   $a := \log_f(g^d)$ 
   $z_{1,j} := \mathcal{I}.\text{DDLog}(\text{pp}, 1, g^{\llbracket \Delta_j d \rrbracket_1^{(\mathbb{Z})}}) \ (\forall j \in [n])$ 
   $z_1 := (z_{1,j})_{j \in [n]} + \mathcal{F}_{k_{\text{WPRF}}}(r^{(i)}) \bmod t$ 
   $z_{0,j} := \mathcal{I}.\text{DDLog}(\text{pp}, 0, g^{\llbracket \Delta_j d \rrbracket_0^{(\mathbb{Z})}}) \ (\forall j \in [n])$ 
   $z_0 := (z_{0,j})_{j \in [n]} + \mathcal{F}_{k_{\text{WPRF}}}(r^{(i)}) \bmod t$ 
   $\mathbf{y}_0 := (a, z_0); \mathbf{y}_1 := (\Delta, z_1)$ 
Return  $b' \leftarrow \mathcal{A}((\mathbf{y}_0^{(i)}, \mathbf{y}_1^{(i)})_{i \in [N]})$ 

```

$H_1(\lambda)$:

```

(pp, |H|) ← PCFmVOLE.Setup(1λ)
For  $i \in [N]$ :  $r^{(i)} \xleftarrow{\$} \{0, 1\}^{\ell_r}$ 
 $(\perp, \Delta) \leftarrow \mathcal{A}(1^\lambda, \text{pp}, (r^{(i)})_{i \in [N]})$ 
 $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, |H|, (\perp, \Delta))$ 
For  $i \in [N]$ :
   $g := \mathcal{I}.\text{SampleG}^{r^{(i)}}(\text{pp})$ 
   $a := \log_f(\pi_F(g))$ 
   $z_{1,j} := \mathcal{I}.\text{DDLog}(\text{pp}, 1, g^{\llbracket \Delta_j d \rrbracket_1^{(\mathbb{Z})}}) \ (\forall j \in [n])$ 
   $z_1 := (z_{1,j})_{j \in [n]} + \mathcal{F}_{k_{\text{WPRF}}}(r^{(i)}) \bmod t$ 
   $z_0 := z_1 - a\Delta \bmod t$ 
   $\mathbf{y}_0 := (a, z_0); \mathbf{y}_1 := (\Delta, z_1)$ 
Return  $b' \leftarrow \mathcal{A}((\mathbf{y}_0^{(i)}, \mathbf{y}_1^{(i)})_{i \in [N]})$ 

```

Experiment 11.5: Hybrids for the Security of $\text{PCF}^{\text{mVOLE}} (\sigma = 0)$

$H_0(\lambda)$:	$H_1(\lambda)$:
$(\text{pp}, H) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Setup}(1^\lambda)$	$(\text{pp}, H) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Setup}(1^\lambda)$
For $i \in [N]$: $r^{(i)} \xleftarrow{\$} \{0, 1\}^{\ell_r}$	For $i \in [N]$: $r^{(i)} \xleftarrow{\$} \{0, 1\}^{\ell_r}$
$(\perp, \Delta) \leftarrow \mathcal{A}(1^\lambda, \text{pp}, (r^{(i)})_{i \in [N]})$	$(\perp, \Delta) \leftarrow \mathcal{A}(1^\lambda, \text{pp}, (r^{(i)})_{i \in [N]})$
$(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, H , (\perp, \Delta))$	$(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, H , (\perp, \Delta))$
For $i \in [N]$:	For $i \in [N]$:
$\mathbf{y}_1 := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_1}(r^{(i)})$	$(a, z_0) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_0}(r^{(i)})$
Return $b' \leftarrow \mathcal{A}(k_0, (\mathbf{y}_1^{(i)})_{i \in [N]})$	$\mathbf{y}_1 := (\Delta, a\Delta + z_0 \bmod t)$
	Return $b' \leftarrow \mathcal{A}(k_0, (\mathbf{y}_1^{(i)})_{i \in [N]})$

Experiment 11.6: Hybrids for the Security of $\text{PCF}^{\text{mVOLE}} (\sigma = 1)$

$H_0(\lambda)$:	$H_1(\lambda)$:
$(\text{pp}, H) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Setup}(1^\lambda)$	$(\text{pp}, H) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Setup}(1^\lambda)$
For $i \in [N]$: $r^{(i)} \xleftarrow{\$} \{0, 1\}^{\ell_r}$	For $i \in [N]$: $r^{(i)} \xleftarrow{\$} \{0, 1\}^{\ell_r}$
$(\perp, \Delta) \leftarrow \mathcal{A}(1^\lambda, \text{pp}, (r^{(i)})_{i \in [N]})$	$(\perp, \Delta) \leftarrow \mathcal{A}(1^\lambda, \text{pp}, (r^{(i)})_{i \in [N]})$
$(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, H , (\perp, \Delta))$	$(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, H , (\perp, \Delta))$
For $i \in [N]$:	For $i \in [N]$:
$(a, z_0) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_0}(r^{(i)})$	$g := \mathcal{I}.\text{SampleGr}^{r^{(i)}}(\text{pp})$
$\mathbf{y}_0 := (a, z_0)$	$a := \log_f(\pi_F(g))$
Return $b' \leftarrow \mathcal{A}(k_1, (\mathbf{y}_0^{(i)})_{i \in [N]})$	$(\Delta, z_1) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_1}(r^{(i)})$
	$z_0 := z_1 - a\Delta \bmod t$
	$\mathbf{y}_0 := (a, z_0)$
	Return $b' \leftarrow \mathcal{A}(k_1, (\mathbf{y}_0^{(i)})_{i \in [N]})$

□

11.2 PCF for Random 1-out-of- t Oblivious Transfer

In [IKNP03], Ishai et al. construct an OT extension protocol (known as IKNP OT extension) that extends a small number of base OTs into a large number of OTs, using only symmetric-key primitives. The protocol first generates a large number of $(2, \lambda)$ -mVOLE samples $((a^{(i)}, z_0^{(i)}), z_1^{(i)})$.¹⁰ This part involves the base OTs, a PRG, and some communication. The obtained mVOLE samples can then be locally converted to 1-out-of-2 OT samples by applying a random oracle $\mathcal{H}$ as follows: Party P_0 computes $m_{a^{(i)}}^{(i)} := \mathcal{H}(z_0^{(i)})$ and P_1 (who additionally knows $\Delta \in \{0, 1\}^\lambda$) computes $m_0^{(i)} := \mathcal{H}(z_1^{(i)})$, $m_1^{(i)} := \mathcal{H}(z_1^{(i)} - \Delta)$. Here, the random oracle can be replaced by a CRHF, i.e., a function $\mathcal{H}$ such that, roughly speaking, $\mathcal{F}_s(\mathbf{x}) := \mathcal{H}(\mathbf{x} \oplus \mathbf{s})$ is a WPRF.

The IKNP OT extension protocol was later optimized in [ALSZ13a, KK13] by using a pseudorandom mVOLE batch instead of a truly random one. That is, the long vector $(z_0^{(i)})_i$ is generated using a PRG, thereby saving roughly half of the communication. More recently, the emergence of PCGs/PCFs has led to *silent* OT extension protocols [BCG⁺19b, BCG⁺19a, BCG⁺20a, OSY21, ADOS22, BCG⁺22] that use a PCG/PCF to locally generate (from short

¹⁰ The original protocol description uses XOR-correlated OT, which is equivalent to $(\mathbb{Z}_2, \mathbb{Z}_2^\lambda)$ -mVOLE.

correlated keys) a large number of mVOLE samples, which are then converted to random OT samples via a CRHF.

Flaw in the Security Proofs of Prior Works. As mentioned in the technical overview, the notion of CRHFs is insufficient to prove the security of all above-mentioned OT extension protocols [ALSZ13a, KK13, BCG⁺19b, BCG⁺19a, BCG⁺20a, OSY21, ADOS22, BCG⁺22] that are based on pseudorandom mVOLE samples. The respective security proofs in [ALSZ13a, ALSZ17, BCG⁺19b] are flawed in the case of a corrupted receiver, since they involve a hybrid argument where pseudorandom samples are replaced by truly random ones (to make them compatible with the CRHF definition) in a context where the adversary knows the PRG seed and can therefore distinguish these hybrids. (The original IKNP protocol from [IKNP03] is not affected.)

There are two ways to modify the protocol so that the proof goes through:

- (1) The first is to use a *strongly* correlation-robust hash function [ALSZ15]. It then suffices for the $z_0^{(i)}$ to be pairwise distinct with overwhelming probability. After our discovery, the ePrint reports [ALSZ13b, ALSZ16] have been updated to use strongly CRHFs.
- (2) The second option is to re-randomize the shares $z_0^{(i)}, z_1^{(i)}$ via a random mask. Such masks can be sampled by one party (since we are in the semi-honest setting) and exchanged with communication linear in the number of samples; however, this would exactly undo the optimization in [ALSZ13a, KK13], and for silent protocols, linear communication is not allowed. Alternatively, a common source of randomness can be used, when available. Note that all above-mentioned silent protocols require such a common source of randomness anyway, either in the form of a random oracle or in the form of the weak PCF's input.

Remark. The OT construction from [BCG⁺20a] uses masks as in option (2), but in a way for which the proof still does not go through. Specifically, they use the same randomness for the masks and for the input to the VOLE PCF, whereas the security proof requires independently sampled masks.

Our Fix. We show that option (2), when applied correctly, yields a secure PCF for random OT. More generally, we are interested in random 1-out-of- t OT for arbitrary $t \geq 2$ (where t is not necessarily prime), so we use the following generalization of correlation-robustness.

Definition 11.7 (($\mathcal{R}, \mathcal{M}$)-Correlation-Robust Hash Function). Let $\mathcal{R} = \mathcal{R}(\lambda)$ be a polynomial-size commutative ring and $\mathcal{M} = \mathcal{M}(\lambda)$ an $\mathcal{R}$ -module. An efficiently computable function $\mathcal{H} : \{0, 1\}^\lambda \times \mathcal{M} \rightarrow \{0, 1\}^\lambda$ is ($\mathcal{R}, \mathcal{M}$)-correlation-robust if, for all $n \in \text{poly}(\lambda)$, the distribution

$$\left\{ \left(\begin{array}{c} t_1, \dots, t_n, r_1, \dots, r_n, \\ (\mathcal{H}(r_1, t_1 - j\Delta), \dots, \mathcal{H}(r_n, t_n - j\Delta))_{j \in \mathcal{R} \setminus \{0\}} \end{array} \right) \mid \begin{array}{l} \Delta, t_1, \dots, t_n \xleftarrow{\$} \mathcal{M}, \\ r_1, \dots, r_n \xleftarrow{\$} \{0, 1\}^\lambda \end{array} \right\} \quad (11.8)$$

is computationally indistinguishable from uniform over $\mathcal{M}^n \times \{0, 1\}^{n\lambda|\mathcal{R}|}$.

When $\mathcal{R} = \mathbb{F}_p$ for prime p , this definition corresponds to $\mathbb{F}_p$ -correlation-robustness from [BCG⁺19b], except that we choose the tweaks $r_1, \dots, r_n$ randomly rather than via a counter, in order to be compatible with the weak PCF setting where the PCF inputs are random.

When generalizing from fields to non-field rings, guessing one of the offsets $j\Delta$ (where $j \in \mathcal{R} \setminus \{0\}$) may become easier, since the action of j on $\mathcal{M}$ is not necessarily bijective, i.e., $j\mathcal{M}$ might be a strict subset of $\mathcal{M}$. Nevertheless, as long as $|j\mathcal{M}|$ is sufficiently large for all $j \in \mathcal{R} \setminus \{0\}$, ($\mathcal{R}, \mathcal{M}$)-CRHFs can exist. Lemma 11.9 shows this in the ROM. In the standard model, it may be possible to securely instantiate ($\mathcal{R}, \mathcal{M}$)-CRHFs based on fixed-key AES, as shown for the special case $\mathcal{R} = \mathbb{Z}_2$ in [GKWY20].

Lemma 11.9 (Random Oracle is $(\mathcal{R}, \mathcal{M})$ -Correlation-Robust). *Let $\mathcal{R} = \mathcal{R}(\lambda)$, $\mathcal{M} = \mathcal{M}(\lambda)$ be such that, for all $j \in \mathcal{R} \setminus \{0\}$, $|j\mathcal{M}|$ is superpolynomial in λ . Then a random oracle $\mathcal{H} : \{0, 1\}^\lambda \times \mathcal{M} \rightarrow \{0, 1\}^\lambda$ is $(\mathcal{R}, \mathcal{M})$ -correlation-robust against all ppt. adversaries with access to $\mathcal{H}$.*

Proof. Let $\mu_j : m \rightarrow jm$ denote the action of j on $\mathcal{M}$. $\ker(\mu_j) = \{m \in \mathcal{M} \mid jm = 0\}$ is a subgroup of $\mathcal{M}$. For every $m \in \mathcal{M}$, it holds that $\mu_j^{-1}(jm) = m + \ker(\mu_j)$, i.e., $\mu_j^{-1}(jm)$ is a coset of $\ker(\mu_j)$. For $y \in \mathcal{M}$, if $y \in j\mathcal{M}$, then $|\mu_j^{-1}(y)| = |\ker(\mu_j)| = |\mathcal{M}|/|j\mathcal{M}|$, otherwise $|\mu_j^{-1}(y)| = 0$.

Let $\mathcal{A}^{\mathcal{H}}$ be a ppt. adversary with access to $\mathcal{H}$. We prove the statement via a sequence of hybrids:

- H_0 corresponds to $\mathcal{A}^{\mathcal{H}}(1^\lambda, \cdot)$ on input drawn from the “real” distribution from Eq. (11.8).
- H_1 is defined like H_0 , except that $\Delta, r_1, \dots, r_n$ are sampled conditioned on the event E that $\forall j \neq j' \in \mathcal{R} \setminus \{0\} : j\Delta \neq j'\Delta$ and that $r_1, \dots, r_n$ are pairwise distinct. Since $j\Delta = j'\Delta \iff \Delta \in \ker(\mu_{j-j'})$, we have $\Pr[\neg E] \leq \sum_{j \neq j'} \frac{1}{|(j-j')\mathcal{M}|} + \binom{n}{2} 2^{-\lambda}$, which is negligible. Hence, $H_0 \approx_s H_1$.

In H_1 , the $n(|\mathcal{R}| - 1)$ arguments $(r_i, t_i - j\Delta)$ to $\mathcal{H}$ are pairwise distinct by E . Hence, the input to the adversary is uniform (up to the conditioning of $r_1, \dots, r_n$ on E), but dependent on $\mathcal{H}$. Let $q \in \text{poly}(\lambda)$ be an upper bound on the number of oracle queries performed by the adversary. W.l.o.g., assume that all oracle queries performed by the adversary use pairwise distinct arguments. We will now, one by one, replace the adversary’s queries to $\mathcal{H}$ with queries to an independent random oracle $\mathcal{H}'$. After q steps, we arrive at a hybrid H_{q+1} where the adversary only invokes $\mathcal{H}'$, and hence, cannot distinguish the input from uniform.

- H_{k+1} for $1 \leq k \leq q$ is defined like H_k , except that, in the k^{th} oracle query, the adversary invokes $\mathcal{H}'$ instead of $\mathcal{H}$. Let (r, y) denote the argument to that query. (r, y) only depends on the input given to the adversary, hence it is independent of Δ . Let E' be the event that $\exists i \in \{1, \dots, n\}, j \in \mathcal{R} \setminus \{0\} : (r, y) = (r_i, t_i - j\Delta)$. Observe that $y = t_i - j\Delta \iff \Delta \in \mu_j^{-1}(t_i - y)$, hence $\Pr[E'] \leq \sum_{j \in \mathcal{R} \setminus \{0\}} \frac{1}{|j\mathcal{M}|}$ is negligible. Conditioned on $\neg E'$, H_{k+1} is distributed identically to H_k , since $\mathcal{H}(r, y)$ is not called elsewhere. Hence, $H_k \approx_c H_{k+1}$.
- H_{q+2} corresponds to $\mathcal{A}^{\mathcal{H}'}(1^\lambda, \cdot)$ on uniform input. In H_{q+1} , the input given to the adversary was already independent of $\mathcal{H}'$ and uniform up to the conditioning of $r_1, \dots, r_n$ on E , so $H_{q+1} \approx_s H_{q+2}$. This concludes the sequence of hybrids, since the naming of $\mathcal{H}'$ does not matter (think of now renaming $\mathcal{H}'$ to $\mathcal{H}$).

Note that $H_0 \approx_c H_{q+2}$ is not necessarily implied by $H_0 \approx_c \dots \approx_c H_{q+2}$ when q is not a constant [FM21]. However, in our case we can deduce $H_0 \approx_c H_{q+2}$, since $q \in \text{poly}(\lambda)$ and, crucially, the distinguishing advantage against any pair (H_k, H_{k+1}) has a common negligible upper bound. $\square$

Lemma 11.10 (Random Oracle is $(\mathbb{Z}_t, \mathbb{Z}_t^n)$ -Correlation-Robust). *Let $t = t(\lambda)$, $n = n(\lambda) \in \mathbb{N}$ such that $\text{spf}(t)^n$ is superpolynomial in λ . Then a random oracle $\mathcal{H} : \{0, 1\}^\lambda \times \mathbb{Z}_t^n \rightarrow \{0, 1\}^\lambda$ is $(\mathbb{Z}_t, \mathbb{Z}_t^n)$ -correlation-robust against all ppt. adversaries with access to $\mathcal{H}$.*

Proof. Let $j \in \mathbb{Z}_t \setminus \{0\}$. By the CRT, there exists a prime power $p^e \mid t$ such that $j \not\equiv 0 \pmod{p^e}$. For $\Delta, \Delta' \in \mathbb{Z}_t$ with $j\Delta = j\Delta'$, it follows that $\Delta \equiv \Delta' \pmod{p}$, thus $|j\mathbb{Z}_t| \geq p$. Therefore, $|j\mathbb{Z}_t^n| \geq p^n \geq \text{spf}(t)^n$ is superpolynomial in λ , so the statement follows by Lemma 11.9. $\square$

Definition 11.11 (Random 1-out-of- t OT Correlation). *Let $\mathcal{P}$ and $t = t(\text{pp})$ be as in Definition 11.1. The random 1-out-of- t OT correlation $(\binom{t}{1}\text{-OT} = (\mathcal{P}, \mathcal{Y}))$ is defined as follows.¹¹*

- $\mathcal{Y}(\text{pp})$: Sample $a \xleftarrow{\$} [t]$ and $m_0, \dots, m_{t-1} \xleftarrow{\$} \{0, 1\}^\lambda$, and output $((a, m_a), (m_0, \dots, m_{t-1}))$.

¹¹ This correlation does not require a master key, hence we avoid specifying $\mathcal{M}$ and omit the mk argument from all algorithms.

The 1-out-of- t OT correlation is reverse-sampleable via the algorithm **RSample** that proceeds on input (pp, σ, y) as follows: If $\sigma = 0$, parse $y =: (a, m') \in [t] \times \{0, 1\}^\lambda$, sample $m_0, \dots, m_{t-1} \xleftarrow{\$} \{0, 1\}^\lambda$ subject to $m_a := m'$, and output $(m_0, \dots, m_{t-1})$; if $\sigma = 1$, parse $y =: (m_0, \dots, m_{t-1}) \in (\{0, 1\}^\lambda)^t$, sample $a \xleftarrow{\$} [t]$, and output (a, m_a) .

In **Construction 11.12**, we present our PCF for random 1-out-of- t OT. The change relative to [BCG⁺19b] is that we add a random mask $\mathbf{r}'$ to the input of $\mathcal{H}$. In our security proof against a semi-honestly corrupted OT receiver, the mask $\mathbf{r}'$ allows us to rely on the indistinguishability guarantee from **Definition 11.7**. The flaw in prior work [BCG⁺19b] was to believe that z_0 (without a mask) is indistinguishable from random due to the pseudorandom $\mathcal{Y}$ -correlated outputs property of PCF^{mVLE} . However, this indistinguishability holds only for outsiders who possess none of the PCF keys.

Construction 11.12: PCF for Random 1-out-of- t OT

Parameters:

- $t = t(\lambda), n = n(\lambda) \in \mathbb{N}$.
- PCF^{mVLE} is a weak PCF for (t, n) -mVLE with input domain $\{0, 1\}^{\ell_r}$ with $\ell_r(\lambda) \geq \lambda$.
- $\mathcal{H}$ is a $(\mathbb{Z}_t, \mathbb{Z}_t^n)$ -CRHF.

$\text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$:

Output $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{mVLE}}.\text{Setup}(1^\lambda)$.

$\text{PCF}^{\text{OT}}.\text{Gen}(\text{pp}, \text{td})$:

1. Sample $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$.
2. Output $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVLE}}.\text{Gen}(\text{pp}, \text{td}, (\perp, \Delta))$.

$\text{PCF}^{\text{OT}}.\text{Eval}(\text{pp}, \sigma, k_\sigma, (r, \mathbf{r}') \in \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n)$:

If $\sigma = 0$:

1. Compute $(a, z_0) := \text{PCF}^{\text{mVLE}}.\text{Eval}_{k_0}(r)$.
2. Output $(a, \mathcal{H}(r, z_0 + \mathbf{r}'))$.

If $\sigma = 1$:

1. Compute $(\Delta, z_1) := \text{PCF}^{\text{mVLE}}.\text{Eval}_{k_1}(r)$.
2. Output $(\mathcal{H}(r, z_1 + \mathbf{r}' - j \cdot \Delta))_{j \in \mathbb{Z}_t}$.

Theorem 11.13. *Let $t, n, \text{PCF}^{\text{mVLE}}, \mathcal{H}$ be as in **Construction 11.12**. Then PCF^{OT} is a weak PCF for $\binom{t}{1}$ -OT.*

Proof. We will prove the two properties from **Definition 10.2** in turn. Since $\binom{t}{1}$ -OT has no master key, $\mathcal{A}$ has no choice in its first invocation, so we may present the hybrids as the distributions of its view across both invocations.

Pseudorandom $\mathcal{Y}$ -Correlated Outputs. We prove this property by a sequence of hybrids shown in **Experiment 11.14**. H_0 corresponds to the real distribution, i.e., $H_0 \equiv \text{Game}_{\mathcal{A}, 1}^{\text{pr}}$. We have inlined the definitions of $\text{PCF}^{\text{OT}}.\text{Gen}$ and $\text{PCF}^{\text{OT}}.\text{Eval}$. $H_0 \approx_c H_1$ holds by the *pseudorandom $\mathcal{Y}$ -correlated outputs* property of PCF^{mVLE} . $H_1 \approx_c H_2$ holds by the $(\mathbb{Z}_t, \mathbb{Z}_t^n)$ -correlation-robustness (cf. **Definition 11.7**) of $\mathcal{H}$. Note that, in H_2 , we did not replace $m_0 := \mathcal{H}(r^{(i)}, z_1)$ with uniform randomness, since **Definition 11.7** only applies to the other messages m_j where $j \neq 0$. However, we can write $z_1 + \mathbf{r}' = (z_1 + \mathbf{r}' + \Delta) - 1 \cdot \Delta$ where $z_1 + \mathbf{r}' + \Delta$ is uniformly random, so we can use the correlation-robustness argument a second time to deduce $H_2 \approx_c H_3$. The latter hybrid corresponds exactly to the ideal distribution, i.e., $H_3 \equiv \text{Game}_{\mathcal{A}, 0}^{\text{pr}}$.

Security. We prove this property separately for $\sigma = 0, 1$.

For $\sigma = 0$, we present a sequence of hybrids in [Experiment 11.15](#). H_0 corresponds to the real distribution, i.e., $H_0 \equiv \text{Game}_{\mathcal{A},0,1}^{\text{sec}}$. We have inlined the definitions of $\text{PCF}^{\text{OT}}.\text{Gen}$ and $\text{PCF}^{\text{OT}}.\text{Eval}$. $H_0 \approx_c H_1$ holds by the *security* of $\text{PCF}^{\text{mVOLE}}$. Note that, in H_1 , k_0 is generated from the canonical master-key share $\widehat{\text{mk}}_1 = \mathbf{0}$ in place of Δ , and hence is independent of Δ . H_1 is distributed identically to H_2 : We only replaced $\mathbf{r}'$ with $\mathbf{r}' - \mathbf{z}_0$ (in every iteration), which doesn't change the distribution, since $\mathbf{r}'$ is uniformly distributed and independent of $\mathbf{z}_0$. This replacement makes the inputs to $\mathcal{H}$ compatible with the CRHF definition. That is, in H_2 , $\mathbf{r}'$ and Δ are chosen uniformly and Δ enters the view only through $\mathcal{H}$, so $H_2 \approx_c H_3$ holds by the $(\mathbb{Z}_t, \mathbb{Z}_t^n)$ -correlation-robustness of $\mathcal{H}$. In the depiction of H_3 , we also reverted the previous replacement of $\mathbf{r}'$, i.e., we replaced $\mathbf{r}' - \mathbf{z}_0$ with $\mathbf{r}'$. Finally, we define H_4 (which we omit from [Experiment 11.15](#) for brevity) like H_3 , except that k_0 is again generated from $(\perp, \Delta)$, as in $\text{PCF}^{\text{OT}}.\text{Gen}$. $H_3 \approx_c H_4$ holds by the security of $\text{PCF}^{\text{mVOLE}}$ once more. In this way, H_4 corresponds exactly to the ideal distribution, i.e., $H_4 \equiv \text{Game}_{\mathcal{A},0,0}^{\text{sec}}$.

For $\sigma = 1$, we present a sequence of hybrids in [Experiment 11.16](#). This case is easier, since it does not rely on the correlation-robustness of $\mathcal{H}$. H_0 corresponds to the real distribution, i.e., $H_0 \equiv \text{Game}_{\mathcal{A},1,1}^{\text{sec}}$. $H_0 \approx_c H_1$ holds by the *security* of $\text{PCF}^{\text{mVOLE}}$. The latter hybrid is distributed identically to the ideal distribution, i.e., $H_1 \equiv \text{Game}_{\mathcal{A},1,0}^{\text{sec}}$, since a is drawn uniformly at random in H_1 .

Experiment 11.14: Hybrids for the $\mathcal{Y}$ -Correlated Outputs of PCF^{OT}

$H_0(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, \text{td}, (\perp, \Delta))$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $(a, z_0) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_0}(r)$ $(\Delta, z_1) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_1}(r)$ $\mathbf{y}_0 := (a, \mathcal{H}(r, z_0 + \mathbf{r}'))$ $\mathbf{y}_1 := (\mathcal{H}(r, z_1 + \mathbf{r}' - j \cdot \Delta))_{j \in [t]}$ Return $(\text{pp}, (r^{(i)}, \mathbf{y}_0^{(i)}, \mathbf{y}_1^{(i)})_{i \in [N]})$	$H_1(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $a \xleftarrow{\$} \mathbb{Z}_t$; $\mathbf{z}_1 \xleftarrow{\$} \mathbb{Z}_t^n$ $\mathbf{z}_0 := \mathbf{z}_1 - a \cdot \Delta \bmod t$ $\mathbf{y}_0 := (a, \mathcal{H}(r, z_0 + \mathbf{r}'))$ $\mathbf{y}_1 := (\mathcal{H}(r, z_1 + \mathbf{r}' - j \cdot \Delta))_{j \in [t]}$ Return $(\text{pp}, (r^{(i)}, \mathbf{y}_0^{(i)}, \mathbf{y}_1^{(i)})_{i \in [N]})$
$H_2(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $a \xleftarrow{\$} \mathbb{Z}_t$; $\mathbf{z}_1 \xleftarrow{\$} \mathbb{Z}_t^n$ $m_0 := \mathcal{H}(r, z_1 + \mathbf{r}')$ $m_1, \dots, m_{t-1} \xleftarrow{\$} \{0, 1\}^\lambda$ $\mathbf{y}_0 := (a, m_a)$; $\mathbf{y}_1 := (m_j)_{j \in [t]}$ Return $(\text{pp}, (r^{(i)}, \mathbf{y}_0^{(i)}, \mathbf{y}_1^{(i)})_{i \in [N]})$	$H_3(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $a \xleftarrow{\$} \mathbb{Z}_t$ $m_0 \xleftarrow{\$} \{0, 1\}^\lambda$ $m_1, \dots, m_{t-1} \xleftarrow{\$} \{0, 1\}^\lambda$ $\mathbf{y}_0 := (a, m_a)$; $\mathbf{y}_1 := (m_j)_{j \in [t]}$ Return $(\text{pp}, (r^{(i)}, \mathbf{y}_0^{(i)}, \mathbf{y}_1^{(i)})_{i \in [N]})$

Experiment 11.15: Hybrids for the Security of PCF^{OT} ($\sigma = 0$)

$H_0(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, \text{td}, (\perp, \Delta))$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $(\Delta, z_1) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_1}(r)$ $\mathbf{y}_1 := (\mathcal{H}(r, z_1 + \mathbf{r}' - j \cdot \Delta))_{j \in [t]}$ Return $(\text{pp}, k_0, (r^{(i)}, \mathbf{r}'^{(i)}, \mathbf{y}_1^{(i)}))_{i \in [N]}$	$H_1(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, \text{td}, (\perp, \mathbf{0}))$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $(a, z_0) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_0}(r)$ $z_1 := a\Delta + z_0$ $\mathbf{y}_1 := (\mathcal{H}(r, z_1 + \mathbf{r}' - j \cdot \Delta))_{j \in [t]}$ Return $(\text{pp}, k_0, (r^{(i)}, \mathbf{r}'^{(i)}, \mathbf{y}_1^{(i)}))_{i \in [N]}$
$H_2(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, \text{td}, (\perp, \mathbf{0}))$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $(a, z_0) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_0}(r)$ $\mathbf{y}_1 := (\mathcal{H}(r, \mathbf{r}' - (j - a) \cdot \Delta))_{j \in [t]}$ Return $(\text{pp}, k_0, (r^{(i)}, \mathbf{r}'^{(i)} - z_0^{(i)}, \mathbf{y}_1^{(i)}))_{i \in [N]}$	$H_3(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, \text{td}, (\perp, \mathbf{0}))$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $(a, z_0) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_0}(r)$ $m_0, \dots, m_{t-1} \xleftarrow{\$} \{0, 1\}^\lambda$ $m_a := \mathcal{H}(r, \mathbf{r}')$ $\mathbf{y}_1 := (m_j)_{j \in [t]}$ Return $(\text{pp}, k_0, (r^{(i)}, \mathbf{r}'^{(i)} - z_0^{(i)}, \mathbf{y}_1^{(i)}))_{i \in [N]}$

Experiment 11.16: Hybrids for the Security of PCF^{OT} ($\sigma = 1$)

$H_0(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, \text{td}, (\perp, \Delta))$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $(a, z_0) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_0}(r)$ $\mathbf{y}_0 := (a, \mathcal{H}(r, z_0 + \mathbf{r}'))$ Return $(\text{pp}, k_1, (r^{(i)}, \mathbf{r}'^{(i)}, \mathbf{y}_0^{(i)}))_{i \in [N]}$	$H_1(\lambda)$: $\Delta \xleftarrow{\$} \mathbb{Z}_t^n$ $(\text{pp}, \text{td}) \leftarrow \text{PCF}^{\text{OT}}.\text{Setup}(1^\lambda)$ $(k_0, k_1) \leftarrow \text{PCF}^{\text{mVOLE}}.\text{Gen}(\text{pp}, \text{td}, (\perp, \Delta))$ For $i \in [N]$: $(r, \mathbf{r}') \xleftarrow{\$} \{0, 1\}^{\ell_r} \times \mathbb{Z}_t^n$ $(\Delta, z_1) := \text{PCF}^{\text{mVOLE}}.\text{Eval}_{k_1}(r)$ $a \xleftarrow{\$} \mathbb{Z}_t; z_0 := z_1 - a\Delta$ $\mathbf{y}_0 := (a, \mathcal{H}(r, z_0 + \mathbf{r}'))$ Return $(\text{pp}, k_1, (r^{(i)}, \mathbf{r}'^{(i)}, \mathbf{y}_0^{(i)}))_{i \in [N]}$
--	---

□

11.3 Public-Key PCFs

In a *public-key PCF*, parties can locally generate a public–secret key pair and, after the public keys have been exchanged, any two parties can locally combine their own secret key with the other party’s public key to derive correlated PCF keys. This enables a single-round PCF key setup.

Our PCFs $\text{PCF}^{\text{mVOLE}}$ and PCF^{OT} can be upgraded to public-key PCFs following the technique from [OSY21]. This technique uses a non-interactive VOLE (NIVOLE) protocol,

i.e., a single-round protocol that allows two parties to compute secret shares of the product of their inputs (over the integers). [OSY21] also shows how to construct a NIVOLE protocol, assuming QR and DCR in a Paillier group given as CRS (generated by a trusted setup). The same CRS can be used by many parties, i.e., the trusted setup is reusable. To set up PCF keys, party P_0 locally runs $\text{PCF}^{\text{mVOLE}}.\text{Setup}$ and derives d from the resulting $|H|$, while party P_1 locally samples Δ ; each of them includes its message, that would be sent in the NIVOLE protocol with input d (resp. Δ), as part of the public key. P_0 additionally includes $\mathbf{pp}$ in the public key. Note that $\mathbf{pp}$ (being sampled by P_0) is not a CRS here; the CRS above for the NIVOLE is a separate one. We refer to [OSY21] for details.

We observe that this technique still works for our PCF (irrespective of the subgroup framework instantiation $\mathcal{I}$ and modulus t used inside the PCF), since the operands for NIVOLE (d and Δ) are simply an integer and an integer vector.

Remark 11.17. The NIVOLE protocol from [OSY21] can be generalized to work over any subgroup framework instantiation that enjoys an efficient DDLog and where H is cyclic and $|H|$ contains only superpolynomially large prime factors. The resulting protocol is secure under the SgI assumption. We observed in Section 4.2 that Paillier_ℓ^2 satisfies this condition, hence we obtain a secure NIVOLE protocol under the DCR assumption (without additionally assuming QR).

11.4 PCF for OLE and Degree-Two Correlations

Definition 11.18 (OLE Correlation). *For a finite commutative ring $\mathcal{R}$, the correlation $\mathcal{R}\text{-OLE} = (\mathcal{P}, \mathcal{Y})$ is defined as follows, where $\mathcal{P} = \{0, 1\}^*$, i.e., the correlation admits all public parameters and $\mathcal{Y}$ ignores them.¹²*

- $\mathcal{Y}(\mathbf{pp})$: Sample $x_0, x_1, z_0 \xleftarrow{\$} \mathcal{R}$, compute $z_1 := z_0 + x_0 \cdot x_1$, and output $((x_0, z_0), (x_1, z_1))$. (Observe that z_0, z_1 are subtractive shares of $x_0 \cdot x_1$.)

Following the blueprint by [OSY21], one obtains a PCG (resp. PCF) for $\mathcal{R}\text{-OLE}$ (or any additive degree-two correlation) over $\mathcal{R} = \mathbb{Z}_t$ by compressing the setup of many parallel instances of our $\text{PCF}^{\text{mVOLE}}$ via a PCG (resp. PCF). This works as follows: Using a PCG/PCF for VOLE over the integers, the parties generate many samples of the form $((d, \llbracket \Delta_i d \rrbracket_0^{(\mathbb{Z})}), (\Delta_i, \llbracket \Delta_i d \rrbracket_1^{(\mathbb{Z})}))$ for a fixed decryption key $d \in [\tilde{N}]$. Each such sample defines a key pair for $\text{PCF}^{\text{mVOLE}}$ (with master key $\mathbf{mk}_i = (\perp, \Delta_i \bmod t)$), which can then be evaluated on randomness r_i to obtain an OLE sample $((a_i, z_{i,0}), (b_i, z_{i,1}))$ over $\mathbb{Z}_t$ where a_i is determined by r_i and $b_i = \Delta_i \bmod t$. To ensure pseudorandom $\mathcal{Y}$ -correlated outputs, the “ $\Delta_i \bmod t$ ” values must be pseudorandom, but we do not care about the pseudorandomness of Δ_i before reduction modulo t . Multiple such OLE samples can be combined to produce any additive degree-two correlation (in the two-party setting).

Unfortunately, one cannot use $\text{PCF}^{\text{mVOLE}}$ to “bootstrap” its own setup, since $\text{PCF}^{\text{mVOLE}}$ outputs shares over $\mathbb{Z}_t$ but the shares of $\llbracket \Delta_i d \rrbracket$ need to be over the integers. [OSY21] showed that the LPN-based PCG from [BCGI18] and the VDLPN-based PCF from [BCG⁺20a] can be modified to output shares over the integers, and hence, can be used in the above construction. Here, VDLPN can also be replaced by the more recent expand-accumulate LPN (EALPN) problem from [BCG⁺22] for better performance.

PCF for OLE from Subgroup Indistinguishability and Sparse LPN. We show that the state-of-the-art PCF for VOLE from [BCM⁺25], which is based on sparse LPN, can similarly be modified to output shares over the integers. At first, this results in much larger

¹² This correlation does not require a master key, hence we avoid to specify $\mathcal{M}$ and omit the $\mathbf{mk}$ argument from all algorithms.

shares $\llbracket \Delta_i d \rrbracket^{(\mathbb{Z})}$ than using the above-mentioned VDLPN/EALPN-based constructions, but we present an optimization that brings the sparse LPN-based approach on par with them. We also present an additional optimization that further reduces the shares of one of the two parties. Having these shares as small as possible is essential, due to the exponentiation $g^{\llbracket \Delta_i d \rrbracket^{(\mathbb{Z})}}$ in $\text{PCF}^{\text{mVOLE}}.\text{Eval}$, which is typically the bottleneck of the PCF evaluation.

Let $\text{hw}(\mathbf{v})$ denote the Hamming weight of a vector $\mathbf{v}$, i.e., the number of non-zero entries of $\mathbf{v}$. We say $\mathbf{v}$ is $\text{hw}(\mathbf{v})$ -sparse if $\text{hw}(\mathbf{v})$ is small compared to the vector dimension.

Definition 11.19 (Regular Vector/Distribution). *Let $w, m, n \in \mathbb{N}$ with $w \mid n$. A vector $\mathbf{v} \in \mathcal{R}^n$ is said to be w -regular, if $\mathbf{v} \in \{\mathbf{u} \in \mathcal{R}^{n/w} \mid \text{hw}(\mathbf{u}) = 1\}^w$, i.e., $\mathbf{v}$ is a concatenation of w scaled unit vectors of equal length. Further, we define*

- $\text{Reg}_{w,n}(\mathcal{R})$ as the uniform distribution over $\{\mathbf{u} \in \mathcal{R}^{n/w} \mid \text{hw}(\mathbf{u}) = 1\}^w$ and
- $\text{RegularCodeGen}(k, n, m, \mathcal{R})$ as the distribution of $(\mathbf{e}_0 \dots \mathbf{e}_{m-1})^T \in \mathcal{R}^{m \times n}$ where each row $\mathbf{e}_i^T$ is sampled as $\mathbf{e}_i \leftarrow \text{Reg}_{k,n}(\mathcal{R})$.

Definition 11.20 (Sparse LPN). *Let $n = n(\lambda)$ be the secret size, $m = m(\lambda)$ the output size, $k = k(\lambda)$ the matrix sparsity and $w = w(\lambda)$ the noise sparsity.*

The $\mathcal{R}$ -SparseLPN(k, n, m, w) assumption states that

$$(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e}) \approx_c (\mathbf{A}, \mathbf{y}),$$

where $\mathbf{A} \leftarrow \text{RegularCodeGen}(k, n, m, \mathcal{R})$, $\mathbf{s} \xleftarrow{\$} \mathcal{R}^n$, $\mathbf{e} \leftarrow \text{Reg}_{w,m}(\mathcal{R})$, $\mathbf{y} \xleftarrow{\$} \mathcal{R}^m$.

The $\mathbb{Z}_t$ -SparseLPN(k, n, m, w) assumption allows us to generate a long pseudorandom vector $\mathbf{b} \in \mathbb{Z}_t^m$ from a small seed $(\mathbf{s}, \mathbf{e})$. Here, the output length m must be less than wn , since otherwise there is an easy distinguisher [BCGI18], i.e., the output length can be at most quadratic in the seed length. As observed in [BCM⁺25], this limitation can be overcome by applying sparse LPN recursively to obtain a larger and larger output at each level of the recursion. That is, for a chain of random w -sparse vectors $\mathbf{e}^{(1)}, \dots, \mathbf{e}^{(L)}$ of ascending length and random k -sparse matrices of appropriate dimensions, the distribution of

$$\mathbf{b} := \mathbf{A}^{(L)} \cdot \left(\dots \left(\mathbf{A}^{(2)} \cdot (\mathbf{A}^{(1)} \cdot \mathbf{s} + \mathbf{e}^{(1)}) + \mathbf{e}^{(2)} \right) \dots \right) + \mathbf{e}^{(L)} \bmod t \quad (11.21)$$

is pseudorandom over $\mathbb{Z}_t^{m^{(L)}}$, even given all of the involved matrices. This allows us to construct a PCF for VOLE as follows: Party P_0 is given a fixed factor d , party P_1 is given $\mathbf{s}, \mathbf{e}^{(1)}, \dots, \mathbf{e}^{(L)}$, and both parties are given secret shares $\llbracket \mathbf{s}d \rrbracket, \llbracket \mathbf{e}^{(1)}d \rrbracket, \dots, \llbracket \mathbf{e}^{(L)}d \rrbracket$. Here, the long but sparse vectors $\llbracket \mathbf{e}^{(1)}d \rrbracket, \dots, \llbracket \mathbf{e}^{(L)}d \rrbracket$ are compressed via standard distributed point functions (DPFs) [BGI16b]. By sampling the matrices from a random oracle and using the linearity of the secret sharing scheme, the parties can compute shares $\llbracket \mathbf{b}d \rrbracket$. In fact, as shown in [BCM⁺25], they can compute any entry of $\llbracket \mathbf{b}d \rrbracket$ on-demand with $\mathcal{O}(k^L)$ DPF evaluations, which makes this a PCF.

In [BCM⁺25], all shares are in the same domain where the sparse LPN assumption is assumed, i.e., $\mathbb{Z}_t$ in our case. To be compatible with the above application, we instead have d in the range $[\hat{N}]$ and sample the shares $\llbracket \mathbf{s}d \rrbracket^{(\mathbb{Z})}, \llbracket \mathbf{e}^{(1)}d \rrbracket^{(\mathbb{Z})}, \dots, \llbracket \mathbf{e}^{(L)}d \rrbracket^{(\mathbb{Z})}$ over the integers in the range $[2^\kappa \hat{N}t]$ for statistical security parameter κ . (This is compatible with above-mentioned DPFs). The sparse LPN-related distributions are unchanged, i.e., the matrices $\mathbf{A}^{(i)}$ and vectors $\mathbf{s}, \mathbf{e}^{(i)}$ (for $i \in \{1, \dots, L\}$) still have coefficients from $[t]$. The parties can then compute (entries of) shares $\llbracket \Delta d \rrbracket^{(\mathbb{Z})}$ where Δ is defined like $\mathbf{b}$ from Eq. (11.21) but with all computations done over the integers.

As mentioned above, we want that $\Delta \bmod t$ is pseudorandom (but we do not care about Δ before reduction modulo t). This holds by the sparse LPN assumption, since $\Delta \bmod t = \mathbf{b}$.

Our Optimizations. In the straightforward construction above, the shares $\llbracket \Delta d \rrbracket^{(\mathbb{Z})}$ are bounded by $\approx 2^\kappa \hat{N} t (tk)^L$, since each multiplication by a k -sparse matrix with entries in $[t]$ contributes with a factor of tk .

Our first optimization reduces the magnitude of the final shares by computing the matrix products first and reducing them modulo t . That is, define $\mathbf{A}^{(L..i)} := \mathbf{A}^{(L)} \cdots \mathbf{A}^{(i)} \bmod t$ (for $i \in \{1, \dots, L\}$) and

$$\Delta := \mathbf{A}^{(L..1)} \cdot \mathbf{s} + \mathbf{A}^{(L..2)} \cdot \mathbf{e}^{(1)} + \dots + \mathbf{A}^{(L..L)} \cdot \mathbf{e}^{(L-1)} + \mathbf{e}^{(L)} \quad (11.22)$$

over the integers. Clearly, it still holds that $\Delta \bmod t = \mathbf{b}$. Using Eq. (11.22), the parties can compute shares $\llbracket \Delta d \rrbracket^{(\mathbb{Z})}$ which are bounded by $\approx 2^\kappa \hat{N} t^2 k^L$, i.e., a factor t^{L-1} improvement.

Our second optimization is that party P_0 can reduce its shares $\llbracket \Delta d \rrbracket_0^{(\mathbb{Z})}$ modulo $|G|$, since these shares are only used as exponents in $\text{PCF}^{\text{mVole}}.\text{Eval}$. The order $|G|$ can be given to P_0 without sacrificing security, since $\text{PCF}^{\text{mVole}}$ only requires the SgI assumption to hold for P_1 . This optimization bounds the shares of P_0 by $|G| < \hat{N}$ and can also be applied to the VDLPN/EALPN-based PCFs.

We analyze the effect of these optimizations for a concrete parameter set using the Paillier instantiation of the subgroup framework with a 3072-bit semiprime, statistical security parameter $\kappa = 64$, $L = 4$, and sparsity $k = 9$ for the sparse LPN matrices (in line with the concrete parameters of [BCM⁺25]). With these parameters, our optimizations reduce the share size from 21 581 bits per party to 12 365 bits for party P_1 and 6 144 bits for party P_0 , reducing the exponentiation runtime by $\approx 43\%$ and $\approx 72\%$, respectively.

Topology-Adaptive PCF for Degree-Two Correlations. The PCF for OLE described so far outputs shares of $a_i b_i$ (for any $i \in [m]$, where m is the output size of the sparse LPN-based PCF). As observed in [OSY21], the parties can also compute shares of $a_j b_i$ for any $j \neq i$ by instead using the randomness r_j to sample the ciphertext. This suffices to instantiate the notion of *topology-adaptive PCF* recently introduced in [CDKSS25].

A topology-adaptive PCF is a PCF that allows the parties to specify the correlation at evaluation time, rather than the correlation being fixed at key-generation time. More specifically, the PCF keys implicitly determine a large set of secret pseudorandom values $\{x_i\}_{i \in I}$ (i.e., $i \mapsto x_i$ is a PRF), and the evaluation takes as input (in addition to the PCF key k_σ) a circuit $\mathcal{C}$ and indices $i_0, \dots, i_{n-1} \in I$ (where n is the number of inputs of $\mathcal{C}$), and outputs shares of $\mathcal{C}(x_{i_0}, \dots, x_{i_{n-1}})$.

Using above observation, we obtain a topology-adaptive PCF for the class of degree-two correlations (i.e., correlations described by a circuit $\mathcal{C}$ of multiplicative depth one) by defining $I := [m]$ and $x_i := a_i + b_i$. The parties can compute shares of x_i for any $i \in I$, since party P_0 can compute a_i and P_1 can compute b_i . Moreover, they can compute shares of $x_i x_j$ via $\llbracket x_i x_j \rrbracket_0 := -a_i a_j + \llbracket a_i b_j \rrbracket_0 + \llbracket a_j b_i \rrbracket_0$ and $\llbracket x_i x_j \rrbracket_1 := b_i b_j + \llbracket a_i b_j \rrbracket_1 + \llbracket a_j b_i \rrbracket_1$. This suffices to compute shares of all terms that may occur in a degree-two correlation.

11.5 Key Sizes and Runtimes

We evaluated our PCFs under the Joye–Libert instantiation with every supported prime-power modulus (i.e., every prime-power t for which we can compute the t^{th} power residue symbol), and under the Paillier instantiation. In all cases, we use a 3072-bit RSA semiprime, statistical security $\kappa = 64$, and computational security $\lambda = 128$. Table 11.23 presents the key sizes (including those of the public-key variant, cf. Section 11.3), Table 11.24 presents the measured runtimes per evaluation, and Table 11.25 presents the key generation and key derivation runtimes in the public-key variant. In those tables, n denotes the rank of the underlying module VOLE correlation, which for $\binom{t}{1}$ -OT is $\lceil \lambda / \log_2 \text{spf}(t) \rceil$ by Lemma 11.10.

We observe that the cost of evaluating the power residue symbol (inside **SampleG**) grows quickly with t : it costs little at $t = 2$ (where it is the Jacobi symbol), and it dominates the entire evaluation at larger values of t , making them infeasible in practice.

Table 11.23. Key sizes of our PCFs. The OLE rows use the sparse LPN parameters of [BCM⁺25], targeting 2^{30} queries.

Correlation	Instantiation	Assumption	n	Public key*(KiB)		Derived key (KiB)		
				$\mathbf{pk}_0$	$\mathbf{pk}_1$	k_0	k_1	
VOLE over $\mathbb{Z}_t$	Joye–Libert	SgI	any $t^\dagger$	1	2.7	1.5	1.5	1.1
	Paillier	DCR	$t = N$	1	6.0	4.5	2.3	1.5
$\binom{t}{1}$ -OT	Joye–Libert	SgI + CRHF	$\text{spf}(t) = 2$	128	2.7	147.4	50.3	49.9
			$\text{spf}(t) = 3$	81	2.7	93.4	32.1	31.8
			$\text{spf}(t) = 5$	56	2.7	64.7	22.6	22.2
			$\text{spf}(t) = 7$	46	2.7	53.2	18.8	18.4
			$\text{spf}(t) = 11$	38	2.7	44.0	15.7	15.3
			$\text{spf}(t) = 13$	35	2.7	40.6	14.5	14.2
			$\text{spf}(t) = 17$	32	2.7	37.1	13.4	13.0
			$\text{spf}(t) = 19$	31	2.7	36.0	13.0	12.6
OLE over $\mathbb{Z}_t$	Joye–Libert	SgI + sparse LPN	any $t^\dagger$	–	–	–	16.8 MiB	17.1 MiB
	Paillier	DCR + sparse LPN	$t = N$	–	–	–	31.8 MiB	62.0 MiB

^{*} Party P_0 publishes the group parameters and one NIVOLE message, whereas party P_1 publishes n of them; both also publish one key exchange message for the WPRF key k_{WPRF} . The public-key variant additionally relies on DCR in the Paillier group used in the NIVOLE protocol (cf. [Remark 11.17](#)).

[†] Every supported t gives the same key sizes.

Table 11.24. Measured evaluation times of our PCFs. Each number is the mean of five evaluations, measured single-threaded on a standard laptop (Intel Core Ultra 7 258V).

Instantiation	t	Setup* (ms)	SampleG† (ms)	VOLE Eval (ms)		$n^\ddagger$	$\binom{t}{1}$ -OT Eval (ms)	
				P_0	P_1		P_0	P_1
Joye–Libert	2	0.83	0.736	4.82	2.82	128	69.4	67.3
	3	57.5	9.48	14.0	11.6	81	64.5	59.9
	4	57.9	9.51	13.6	11.5	128	80.0	75.8
	5	128	26.4	30.2	28.0	56	68.8	64.8
	7	318	52.1	56.1	54.1	46	88.1	85.1
	8	177	23.9	27.8	26.0	128	97.2	95.8
	9	312	56.7	61.2	58.7	81	112	110
	11	1 292	211	214	213	38	244	242
	13	4 042	344	346	346	35	376	373
	16	639	130	135	133	128	205	204
	17	16 882	749	758	752	32	780	777
	19	26 968	1 211	1 210	1 213	31	1 239	1 238
	25	45 088	1 964	1 965	1 961	56	1 999	1 997
	27	26 530	1 149	1 154	1 151	81	1 203	1 203
	32	26 474	1 943	1 954	1 946	128	2 019	2 018
Paillier	N	1.13	0.238	34.9	21.0	–	–	–

* Precomputation for the given parameters, done once before any evaluation. It grows with t because of the generator of ρ_m and the element x_1 that our sampling algorithm needs (cf. [Section 5.2](#)).

† Included in the Eval times, since every evaluation begins by sampling g .

‡ The rank is 1 for VOLE, and is listed here only for $\binom{t}{1}$ -OT.

Table 11.25. Measured runtime of the public-key setup of [Section 11.3](#) on a standard laptop (Intel Core Ultra 7 258V). Each party generates and publishes one public key, and then derives its PCF key from the other party’s public key.

Correlation	Instantiation	n	$\log \tilde{N}^*$	Key generation (ms)		Key derivation (ms)	
				P_0	P_1	P_0	P_1
VOLE over $\mathbb{Z}_t$	Joye–Libert	any t	1	3 137	33.2	17.7	18.2
	Paillier	$t = N$	1	9 280	745	454	458
$\binom{t}{1}$ -OT	Joye–Libert	$\text{spf}(t) = 2$	128	3 137	33.3	486	2 059
		$\text{spf}(t) = 3$	81	3 134	33.2	348	1 298
		$\text{spf}(t) = 5$	56	3 133	33.3	278	898
		$\text{spf}(t) = 7$	46	3 134	33.3	236	743
		$\text{spf}(t) = 11$	38	3 133	33.2	213	612
		$\text{spf}(t) = 13$	35	3 134	33.2	199	563
		$\text{spf}(t) = 17$	32	3 133	33.4	196	516
		$\text{spf}(t) = 19$	31	3 133	33.2	183	499

* The modulus of the CRS that the NIVOLE runs over.

Acknowledgments. We thank Geoffroy Couteau, Ralf Küsters, Pascal Reisert, and the anonymous reviewers for their comments. This research was supported by Advantest as part of the Graduate School “Intelligent Methods for Test and Reliability” (GS-IMTR) at the University of Stuttgart. The implementation used to obtain the runtimes in [Tables 9.8](#), [11.24](#) and [11.25](#) was written with the assistance of a generative AI tool, Claude Opus 5 (Anthropic).

References

- ADOS22. Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Cham, August 2022.
- ALSZ13a. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 2013*, pages 535–548. ACM Press, November 2013.
- ALSZ13b. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. Cryptology ePrint Archive, Report 2013/552, 2013.
- ALSZ15. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 673–701. Springer, Berlin, Heidelberg, April 2015.
- ALSZ16. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. Cryptology ePrint Archive, Report 2016/602, 2016.
- ALSZ17. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 30(3):805–858, July 2017.
- Bar86. David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1 . In *18th ACM STOC*, pages 1–5. ACM Press, May 1986.
- BCG⁺17. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.
- BCG⁺19a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.
- BCG⁺19b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.
- BCG⁺20a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.
- BCG⁺20b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. Cryptology ePrint Archive, Report 2020/1417, 2020.
- BCG⁺22. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, Cham, August 2022.
- BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.
- BCM⁺25. Lennart Braun, Geoffroy Couteau, Kelsey Melissaris, Mahshid Riahinia, and Elahe Sadeghi. Fast pseudorandom correlation functions from sparse LPN. In *ASIACRYPT 2025, Part VII*, volume 16251 of *LNCS*, pages 439–473. Springer, Singapore, December 2025.
- BDK⁺11. Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 1–20. Springer, Berlin, Heidelberg, August 2011.
- Ben94. Josh Benaloh. Dense probabilistic encryption. In *SAC 1994*, pages 120–128, 1994.
- BG10. Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In *CRYPTO 2010*, volume 6223 of *LNCS*, pages 1–20. Springer, Berlin, Heidelberg, August 2010.
- BGI16a. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Berlin, Heidelberg, August 2016.
- BGI16b. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

- BGI17. Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Cham, April / May 2017.
- BGSJN21. Eric Brier, Rémi Géraud-Stewart, Marc Joye, and David Naccache. Primary elements in cyclotomic fields with applications to power residue symbols, and more. Cryptology ePrint Archive, Report 2021/1106, 2021.
- BHHO08. Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, Berlin, Heidelberg, August 2008.
- BKS19. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Cham, May 2019.
- BN19. Eric Brier and David Naccache. The thirteenth power residue symbol. Cryptology ePrint Archive, Report 2019/1176, 2019.
- CDKSS25. Geoffroy Couteau, Srinivas Devadas, Alexander Koch, and Sacha Servan-Schreiber. Pseudorandom correlation functions for garbled circuits. In *TCC 2025, Part II*, volume 16269 of *LNCS*, pages 462–497. Springer, Cham, December 2025.
- CL15. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA 2015*, volume 9048 of *LNCS*, pages 487–505. Springer, Cham, April 2015.
- CS10. Perlas C. Caranay and Renate Scheidler. An efficient seventh power residue symbol algorithm. *International Journal of Number Theory*, 06(08):1831–1853, 2010.
- dBP17. Koen de Boer and Carlo Pagano. Calculating the power residue symbol and Ibeta: Applications of computing the group structure of the principal units of a p-adic number field completion. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2017*, pages 117–124, Kaiserslautern, Germany, 2017. ACM.
- DF05. Ivan Damgård and Gudmund Skovbjerg Frandsen. Efficient algorithms for the gcd and cubic residuosity in the ring of Eisenstein integers. *J. Symb. Comput.*, 39(6):643–652, 2005.
- DJ01. Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Berlin, Heidelberg, February 2001.
- DKK18. Itai Dinur, Nathan Keller, and Ohad Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 213–242. Springer, Cham, August 2018.
- ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
- FGJS17. Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith, III. Homomorphic secret sharing from paillier encryption. In *ProvSec 2017*, volume 10592 of *LNCS*, pages 381–399. Springer, Cham, October 2017.
- FM21. Marc Fischlin and Arno Mittelbach. An overview of the hybrid argument. Cryptology ePrint Archive, Report 2021/088, 2021.
- GKWY20. Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, pages 825–841. IEEE Computer Society Press, May 2020.
- GM82. Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982.
- Has26. Sebastian Hasler. Beyond DCR: HSS and PCFs from subgroup indistinguishability. In *ASIACRYPT 2026*, LNCS. Springer, 2026.
- HILL99. Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.
- HR26. Sebastian Hasler and Pascal Reiser. Pseudorandom correlation functions for multiparty Beaver triples from sparse LPN. In *ASIACRYPT 2026*, LNCS. Springer, 2026.
- HRK25. Sebastian Hasler, Pascal Reiser, and Ralf Küsters. Pseudorandom correlation functions from ring-LWR. In *ASIACRYPT 2025, Part VII*, volume 16251 of *LNCS*, pages 401–438. Springer, Singapore, December 2025.
- IKNP03. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Berlin, Heidelberg, August 2003.
- Iwa68. Kenkichi Iwasawa. On explicit formulas for the norm residue symbol. *Journal of the Mathematical Society of Japan*, 20(1-2):151–165, 1968.
- JL13. Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k -th power residue symbols. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 76–92. Springer, Berlin, Heidelberg, May 2013.
- JLNN21. Marc Joye, Oleksandra Lapiha, Ky Nguyen, and David Naccache. The eleventh power residue symbol. *J. Math. Cryptol.*, 15(1):111–122, 2021.
- Joy19. Marc Joye. Evaluating octic residue symbols. Cryptology ePrint Archive, Report 2019/1196, 2019.

- KK13. Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Berlin, Heidelberg, August 2013.
- KOR23. Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy. Two-round stateless deterministic two-party Schnorr signatures from pseudorandom correlation functions. In *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 646–677. Springer, Cham, August 2023.
- Lem95. Franz Lemmermeyer. The Euclidean algorithm in algebraic number fields. *Expositiones Mathematicae*, 13(5):385–416, 1995. Updated version (2004): <http://www.fen.bilkent.edu.tr/~franz/publ/survey.pdf>.
- MM76. John M. Masley and Hugh L. Montgomery. Cyclotomic fields with unique factorization. *Journal für die reine und angewandte Mathematik*, 286/287:248–256, 1976.
- Neu99. Jürgen Neukirch. *Algebraic Number Theory*, volume 322 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1999.
- OSY21. Claudio Orlandi, Peter Scholl, and Sophia Yakubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Cham, October 2021.
- Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT’99*, volume 1592 of *LNCS*, pages 223–238. Springer, Berlin, Heidelberg, May 1999.
- PH78. Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.
- RS21. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Cham.
- Sho05. Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2005.
- SW95. Renate Scheidler and Hugh C. Williams. A public-key cryptosystem utilizing cyclotomic fields. *DCC*, 6(2):117–131, 1995.
- Vos79. Sergei V. Vostokov. Explicit form of the law of reciprocity. *Mathematics of the USSR-Izvestiya*, 13(3):557–588, 1979.
- Wei02. André Weilert. Fast computation of the biquadratic residue symbol. *Journal of Number Theory*, 96(1):133–151, 2002.
- Wil86. Hugh C. Williams. An M^3 public-key encryption scheme. In *CRYPTO’85*, volume 218 of *LNCS*, pages 358–368. Springer, Berlin, Heidelberg, August 1986.
- ZCDL25. Xiaopeng Zhao, Zhenfu Cao, Xiaolei Dong, and Zhusen Liu. Higher residuosity attacks on small RSA subgroup decision problems. In *PKC 2025, Part I*, volume 15674 of *LNCS*, pages 67–87. Springer, Cham, May 2025.

A Power Residue Symbols for Further Prime Powers

AI Disclosure. We found the result of this appendix with the aid of a generative AI tool, Claude Fable 5 (Anthropic). It proposed extending the approach of [BGSJN21] to the prime powers treated here, wrote the scripts behind the computations reported below, and drafted this appendix. We revised its presentation.

In this appendix, we extend the approach of [BGSJN21] for computing the m^{th} power residue symbol to the prime powers $m \in \{17, 19, 25, 27, 32\}$, which yields the corresponding entries of Table 5.1. Together with the algorithms from the literature, this covers every prime power m for which $\mathbb{Z}[\zeta_m]$ has class number one [MM76]. For the remaining prime powers, the ideal ρ_m from Section 5 need not be principal, so the approach below, which computes with elements rather than ideals, does not apply. We state the algorithm for any prime power m of Table 5.1. For $m \in \{9, 16\}$, it is the recursion of [BGSJN21] with an escape step added, which makes its termination provable. For the smaller m , the algorithms cited in Table 5.1 are alternatives.

Throughout, let $m = p^k$, $\mathcal{R} := \mathbb{Z}[\zeta_m]$, and $\omega := 1 - \zeta_m$, which generates the unique prime ideal of $\mathcal{R}$ above p , so that $(p) = (\omega)^e$ for $e := \varphi(m)$. For $p \nmid a$, the cyclotomic unit $u_a := (1 - \zeta_m^a)/(1 - \zeta_m) \equiv a \pmod{\omega}$ is a unit of $\mathcal{R}$. Since $\mathcal{R}/(\omega) \cong \mathbb{F}_p$, every $\alpha \in \mathcal{R}$ coprime to ω satisfies $u_r \alpha \equiv 1 \pmod{\omega}$ for exactly one $r \in \{1, \dots, p-1\}$, where $u_1 = 1$, and we call $u_2, \dots, u_{p-1}$ the normalizing units. For $p = 2$, there are none, since every element coprime to ω is congruent to 1 modulo ω . Let $D := ek + p^{k-1}$. Table A.1 lists e and D for the prime powers $m \geq 9$ of Table 5.1. Every $\alpha \in \mathcal{R}$ with $\alpha \equiv 1 \pmod{\omega^{D+1}}$ is an m^{th} power in the completion of $\mathcal{R}$ at ω [Neu99, Ch. II, §5], so all symbols below depend on their arguments only modulo ω^{D+1} . Every $\alpha \equiv 1 \pmod{\omega}$ can be written as $\alpha \equiv \prod_{i=1}^D (1 + \omega^i)^{e_i} \pmod{\omega^{D+1}}$ with $e_i \in \{0, \dots, p-1\}$, found digit by digit in the ω -adic expansion, and we write $e(\alpha) := (e_1, \dots, e_D)$.

Table A.1. Parameters of the algorithms of this appendix for each prime power m : the degree $e = \varphi(m)$, the precision D , which is the dimension of T and of the U_γ , and the number 3^g of units that Reduce tries, g being the number of a with $2 \leq a < m/2$ and $p \nmid a$. The columns $m = 9$ and $m = 16$ are the cases of [BGSJN21], whose D of 15 and 40 we reproduce.

	$m = 9$	$m = 16$	$m = 17$	$m = 19$	$m = 25$	$m = 27$	$m = 32$
$e = \varphi(m)$	6	8	16	18	20	18	16
D	15	40	17	19	45	63	96
3^g (units per reduction)	9	27	2187	6561	19683	6561	2187

Reciprocity and Complementary Laws. For $\alpha, \beta \in \mathcal{R}$ that are coprime to each other and to ω , the reciprocity law for the m^{th} power residue symbol [Neu99, Ch. VI, §8] expresses $\left(\frac{\alpha}{\beta}\right)_m \left(\frac{\beta}{\alpha}\right)_m^{-1}$ as the product of the Hilbert symbols $(\alpha, \beta)_{\mathfrak{p}}$ over the places $\mathfrak{p}$ of $\mathbb{Q}(\zeta_m)$ dividing m or infinity. Here, the infinite places are complex and contribute nothing, and the only finite one is ω , so the quotient equals the Hilbert symbol $(\alpha, \beta)_{\omega}$ or its inverse, depending on the orientation convention. The Hilbert symbol is bimultiplicative and trivial whenever one argument is congruent to 1 modulo ω^{D+1} – such elements are m^{th} powers in the completion – so it depends only on the classes of its arguments modulo ω^{D+1} and is determined by its

values on the generators $1 + \omega^i$. We therefore define $T \in \mathbb{Z}_m^{D \times D}$ by

$$\zeta_m^{T_{ij}} := \left(\frac{\alpha}{\beta} \right)_m \left(\frac{\beta}{\alpha} \right)_m^{-1} \quad \text{for any coprime } \alpha \equiv 1 + \omega^i \text{ and } \beta \equiv 1 + \omega^j \pmod{\omega^{D+1}},$$

which does not depend on the choice of α and β , since it is the value of the Hilbert symbol, or of its inverse, on the pair $(1 + \omega^i, 1 + \omega^j)$. Writing α and β as products of the generators then gives, for all coprime $\alpha, \beta \equiv 1 \pmod{\omega}$,

$$\left(\frac{\alpha}{\beta} \right)_m = \left(\frac{\beta}{\alpha} \right)_m \cdot \zeta_m^{e(\alpha)^\top T e(\beta)}, \quad (\text{A.2})$$

no matter which exponent vectors represent them. Likewise, for γ a unit of $\mathcal{R}$ or $\gamma = \omega$ and for β coprime to γ and to ω , the same law makes $\left(\frac{\gamma}{\beta} \right)_m$ the Hilbert symbol $(\gamma, \beta)_\omega$ or its inverse. Hence, there is a vector $U_\gamma \in \mathbb{Z}_m^D$ with $\left(\frac{\gamma}{\beta} \right)_m = \zeta_m^{U_\gamma \cdot e(\beta)}$ for all $\beta \equiv 1 \pmod{\omega}$ coprime to γ , namely $\zeta_m^{U_{\gamma,i}} := \left(\frac{\gamma}{\beta} \right)_m$ for any $\beta \equiv 1 + \omega^i \pmod{\omega^{D+1}}$ coprime to γ . We use this for $\gamma = \omega$ and for the normalizing units $u_r, r \in \{2, \dots, p-1\}$.

We obtain T and the vectors U_γ from the definition of the symbol: For each $i \leq D$, we pick an element $\pi_i \equiv 1 + \omega^i \pmod{\omega^{D+1}}$ whose norm is a prime number, so that (π_i) is a prime ideal of degree one and $\left(\frac{\cdot}{\pi_i} \right)_m$ is computed by an exponentiation modulo the norm. Then, for $i \neq j$, T_{ij} is the exponent of $\left(\frac{\pi_i}{\pi_j} \right)_m \left(\frac{\pi_j}{\pi_i} \right)_m^{-1}$, and $U_{\gamma,i}$ is that of $\left(\frac{\gamma}{\pi_i} \right)_m$. For T_{ii} , we use a second prime element $\pi'_i \equiv 1 + \omega^i \pmod{\omega^{D+1}}$ in place of π_i . For odd m , the antisymmetry of T forces $T_{ii} = 0$, but for $m \in \{16, 32\}$ some diagonal entries equal $m/2$. We have carried out this computation for every prime power in [Table 5.1](#), verified that the result does not depend on the choice of the π_i and that T is antisymmetric, and, for $m \in \{9, 16\}$, that it reproduces the tables of [\[BGSJN21\]](#). For $m \in \{17, 19\}$, [Table A.6](#) lists T and the vectors U_γ . For $m \in \{9, 16\}$, [\[BGSJN21\]](#) tabulate the same correction, and for prime $m \leq 13$, they instead normalize so that it vanishes. Explicit formulas for this factor are classical, at least for odd p [\[Iwa68, Vos79\]](#), and it can also be evaluated by computing in a p -adic completion [\[dBP17\]](#); a table, however, is computed once per m and reduces every application of the reciprocity law to evaluating a fixed bilinear form over $\mathbb{Z}_m$.

The Reduction Step. For $\alpha, \beta \in \mathcal{R}$ with $\beta \neq 0$, the procedure $\text{Reduce}(\alpha, \beta)$ computes a remainder of α modulo β as follows. Let u range over the 3^g products $\prod_a u_a^{x_a}$ with $x_a \in \{-1, 0, 1\}$, where a runs over the g integers with $2 \leq a < m/2$ and $p \nmid a$ (cf. [Table A.1](#)). For each u , let $\mu_u := \alpha - u^{-1} \lfloor u\alpha/\beta \rfloor \beta$, where $\lfloor \cdot \rfloor$ rounds each coefficient in the power basis, and output the μ_u of smallest norm. The remainder found this way need not have norm below $\mathcal{N}(\beta)$: $\mathbb{Z}[\zeta_{32}]$ is not norm-Euclidean, so for $m = 32$ some residue classes contain no element of smaller norm at all, and for $m \in \{17, 19, 25, 27\}$ it is unknown whether $\mathbb{Z}[\zeta_m]$ is norm-Euclidean [\[Lem95, Sect. 8\]](#). When the reduction fails, [Algorithm A.3](#) escapes to a different residue class using the set $\mathcal{E} := \{\delta \in \mathcal{R} \mid \delta \equiv 1 \pmod{\omega}, \mathcal{N}(\delta) \text{ is prime}\}$, enumerated by increasing maximum absolute coefficient with ties broken in any fixed way, an order in which every element occurs. Elements of prime norm are common among those with small coefficients, so a suitable δ is found within a few candidates. Which units Reduce tries and how $\mathcal{E}$ is enumerated affects only the runtime of [Algorithm A.3](#), not its output.

The Algorithm. [Algorithm A.3](#) computes the exponent j with $\left(\frac{\alpha}{\beta} \right)_m = \zeta_m^j$. The sampler of [Section 5](#) calls it with $\alpha := x \in \mathbb{Z}$ and β a generator of ρ_m . We describe how to compute this generator at the end of this appendix, and we prove in [Theorem A.4](#) below that [Algorithm A.3](#) terminates with the correct output.

Algorithm A.3: m^{th} Power Residue Symbol

Parameters: the tables T and $(U_\gamma)_{\gamma \in \{u_2, \dots, u_{p-1}, \omega\}}$ and the enumeration $\mathcal{E}$, as defined in this appendix.

Input: $\alpha, \beta \in \mathcal{R}$ coprime to each other, with β coprime to ω .

Output: $j \in \mathbb{Z}_m$ with $\left(\frac{\alpha}{\beta}\right)_m = \zeta_m^j$.

1. Set $j := 0$.
2. If $\beta \not\equiv 1 \pmod{\omega}$: Replace β by $u_r \beta$ for the r with $u_r \beta \equiv 1 \pmod{\omega}$.
3. While β is not a unit:
 - (a) Compute $\mu := \text{Reduce}(\alpha, \beta)$.
 - (b) If $\mathcal{N}(\mu) \geq \mathcal{N}(\beta)$: Let δ be the first element of $\mathcal{E}$ with $\mathcal{N}(\delta) \nmid \mathcal{N}(\beta)$ for which $\delta\mu \equiv 1 \pmod{\beta}$ or $\tilde{\mu} := \text{Reduce}(\delta\mu, \beta)$ satisfies $\mathcal{N}(\tilde{\mu}) < \mathcal{N}(\beta)$. Set $\mu := 1$ in the former case and $\mu := \tilde{\mu}$ in the latter, and set $j := j - \log_{\zeta_m} \left(\frac{\beta}{\delta}\right)_m - e(\delta)^\top Te(\beta)$.
 - (c) Write $\mu = \omega^s \mu'$ with $\mu' \not\equiv 0 \pmod{\omega}$ and set $j := j + s \cdot U_\omega \cdot e(\beta)$.
 - (d) If $\mu' \not\equiv 1 \pmod{\omega}$: Replace μ' by $u_r \mu'$ for the r with $u_r \mu' \equiv 1 \pmod{\omega}$ and set $j := j - U_{u_r} \cdot e(\beta)$.
 - (e) Set $j := j + e(\mu')^\top Te(\beta)$ and $(\alpha, \beta) := (\beta, \mu')$.
4. Output $j \bmod m$.

Theorem A.4. *Let $\alpha, \beta \in \mathcal{R}$ be coprime to each other, with β coprime to ω . Then **Algorithm A.3** terminates on input (α, β) and outputs j with $\left(\frac{\alpha}{\beta}\right)_m = \zeta_m^j$.*

Proof. Write α_0, β_0 for the inputs. Step 2 replaces β by a unit multiple, which changes neither the ideal (β) nor the symbol, and afterwards $\beta \equiv 1 \pmod{\omega}$. We claim that at the start of every iteration of the loop, $\beta \equiv 1 \pmod{\omega}$, α and β are coprime, and

$$\left(\frac{\alpha_0}{\beta_0}\right)_m = \zeta_m^j \left(\frac{\alpha}{\beta}\right)_m. \quad (\text{A.5})$$

Before the first iteration, this holds with $j = 0$. We track Eq. (A.5) through one iteration, where it holds with μ in place of α after Steps 3a and 3b, and with μ' in place of α after Steps 3c and 3d. Step 3a yields $\mu \equiv \alpha \pmod{\beta}$, so $\left(\frac{\alpha}{\beta}\right)_m = \left(\frac{\mu}{\beta}\right)_m$, and μ is coprime to β . If Step 3b runs, then δ is coprime to β , since (δ) is a prime ideal and $(\delta) \mid (\beta)$ would imply $\mathcal{N}(\delta) \mid \mathcal{N}(\beta)$. By multiplicativity, $\left(\frac{\mu}{\beta}\right)_m = \left(\frac{\delta\mu}{\beta}\right)_m \left(\frac{\delta}{\beta}\right)_m^{-1}$. By Eq. (A.2) for δ and β , which are $\equiv 1 \pmod{\omega}$ and coprime to each other and to ω , $\left(\frac{\delta}{\beta}\right)_m = \left(\frac{\beta}{\delta}\right)_m \zeta_m^{e(\delta)^\top Te(\beta)}$, where $\left(\frac{\beta}{\delta}\right)_m$ is evaluated by the definition, δ being a prime of degree one, i.e., by one exponentiation modulo the prime $\mathcal{N}(\delta)$. The step then replaces μ by an element $\equiv \delta\mu \pmod{\beta}$, namely by 1 or by $\tilde{\mu}$, which is hence coprime to β , and subtracts $\log_{\zeta_m} \left(\frac{\beta}{\delta}\right)_m + e(\delta)^\top Te(\beta)$ from j , so that Eq. (A.5) holds with the new μ . Step 3c writes $\mu = \omega^s \mu'$, so $\left(\frac{\mu}{\beta}\right)_m = \left(\frac{\omega}{\beta}\right)_m^s \left(\frac{\mu'}{\beta}\right)_m = \zeta_m^{s \cdot U_\omega \cdot e(\beta)} \left(\frac{\mu'}{\beta}\right)_m$ by the complementary law for ω , and the update of j accounts for the factor; μ' is coprime to β and to ω . If Step 3d runs, then $\left(\frac{\mu'}{\beta}\right)_m = \left(\frac{u_r \mu'}{\beta}\right)_m \left(\frac{u_r}{\beta}\right)_m^{-1} = \left(\frac{u_r \mu'}{\beta}\right)_m \zeta_m^{-U_{u_r} \cdot e(\beta)}$ by the complementary law for u_r , which the update of j accounts for again, and afterwards $\mu' \equiv 1 \pmod{\omega}$. Step 3e applies Eq. (A.2) to μ' and β , both $\equiv 1 \pmod{\omega}$ and coprime to each other and to ω , which gives $\left(\frac{\mu'}{\beta}\right)_m = \left(\frac{\beta}{\mu'}\right)_m \zeta_m^{e(\mu')^\top Te(\beta)}$. After the update of j , Eq. (A.5) therefore holds for the new pair $(\alpha, \beta) = (\beta, \mu')$, whose second component is $\equiv 1 \pmod{\omega}$ and coprime

to the first. The loop ends with β a unit, so $(\beta) = \mathcal{R}$ and $\left(\frac{\alpha}{\beta}\right)_m = 1$, the empty product, and Eq. (A.5) yields $\left(\frac{\alpha_0}{\beta_0}\right)_m = \zeta_m^j$.

For termination, note that every iteration replaces β by μ' with $\mathcal{N}(\mu') \leq \mathcal{N}(\mu) < \mathcal{N}(\beta)$: $\mathcal{N}(\mu) < \mathcal{N}(\beta)$ holds after Step 3a or is established by Step 3b, Step 3c divides the norm by p^s , and Step 3d preserves it. The norms of the successive β are thus strictly decreasing positive integers, so Algorithm A.3 terminates as soon as every search in Step 3b does. Pick $a \in \mathcal{R}$ with $a \equiv \mu^{-1} \pmod{\beta}$ and $a \equiv 1 \pmod{\omega}$, which exists since μ is coprime to β , and β to ω . By the prime ideal theorem for ray classes [Neu99, Ch. VII, §13], infinitely many prime ideals of degree one lie in the ray class of (a) modulo $\beta\omega$, and each of them equals $(ac) = (\delta)$ for some $c \equiv 1 \pmod{\beta\omega}$, i.e., has a generator $\delta \equiv a \pmod{\beta\omega}$. These δ lie in $\mathcal{E}$ and satisfy $\delta\mu \equiv 1 \pmod{\beta}$, and all but finitely many of them satisfy $\mathcal{N}(\delta) \nmid \mathcal{N}(\beta)$, so the enumeration reaches one unless an earlier δ succeeds. $\square$

Runtime. The proof yields no bound on the runtime, which remains heuristic. The same holds for the algorithms of [BGSJN21], whose termination is moreover not proven. In our experiments, no more than about one percent of the reductions escaped, and the first candidate succeeded in every escape. An escape costs one evaluation of $\left(\frac{\beta}{\delta}\right)_m$, an exponentiation modulo the small prime $\mathcal{N}(\delta)$, and one evaluation of the bilinear form, and the first elements of $\mathcal{E}$ can be precomputed together with their exponent vectors $e(\delta)$.

Computing a Generator of ρ_m . The sampler of Section 5 needs a generator of $\rho_m = (N, \zeta_m - f^{t/m})$, which we compute once per $\mathbf{pp}$ as $\gcd(N, \zeta_m - f^{t/m})$ by the Euclidean algorithm with Reduce: starting from $(\alpha, \beta) := (N, \zeta_m - f^{t/m})$, we replace (α, β) by $(\beta, \text{Reduce}(\alpha, \beta))$ until $\beta = 0$ and output α . This is correct whenever it terminates, since each replacement preserves the ideal (α, β) . When the reduction fails, the escape of Step 3b applies without a correction term: for $\delta \in \mathcal{E}$ with $\mathcal{N}(\delta) \nmid \mathcal{N}(\beta)$, δ is coprime to β , so $\gcd(\delta\mu, \beta) = \gcd(\mu, \beta) = \gcd(\alpha, \beta)$, and we replace (α, β) by $(\beta, \text{Reduce}(\delta\mu, \beta))$ for the first such δ for which this remainder has norm below $\mathcal{N}(\beta)$. Here the inputs share a factor, so μ is not invertible modulo β , the termination argument of Theorem A.4 does not apply, and termination of this computation stays heuristic.

Table A.6. The reciprocity matrix $T \in \mathbb{Z}_m^{m \times m}$ and the complementary-law vectors $U_\gamma \in \mathbb{Z}_m^m$, computed as described in this appendix. Row i lists T_{ij} for $j = 1, \dots, m$, followed by $U_{\gamma,i}$ for $\gamma = u_2, \dots, u_{m-1}, \omega$. For $\alpha, \beta \equiv 1 \pmod{\omega}$ that are coprime to each other, $\left(\frac{\alpha}{\beta}\right)_m = \left(\frac{\beta}{\alpha}\right)_m \zeta_m^{\epsilon(\alpha)^\top T \epsilon(\beta)}$, and $\left(\frac{\gamma}{\beta}\right)_m = \zeta_m^{U_\gamma \cdot \epsilon(\beta)}$.

(a) $m = 17$

[illegible]

(b) $m = 19$

T_{ij} for $j = 1, \dots, 19$																			$U_{\gamma,i}$ for $\gamma = u_2, \dots, u_{18}, \omega$																		
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	ω
1	0	16	11	1	5	15	11	9	11	18	1	18	1	18	1	18	1	18	0	8	13	17	9	10	17	16	1	4	6	13	12	17	12	14	15	13	0
2	3	0	3	0	5	0	2	0	17	0	2	0	17	0	2	0	17	0	0	7	3	2	13	10	2	14	16	8	9	0	11	17	9	13	1	16	0
3	8	16	0	12	11	0	3	11	0	16	0	0	3	0	0	16	0	0	0	8	13	17	9	10	17	16	1	4	6	13	12	17	12	14	15	13	0
4	18	0	7	0	5	0	15	0	0	0	4	0	0	0	15	0	0	0	0	3	8	7	11	3	18	1	15	5	9	6	9	16	11	11	5	1	0
5	14	14	8	14	0	0	12	0	5	0	0	0	0	14	0	0	0	0	0	9	17	9	14	13	1	1	10	12	7	11	8	13	12	5	1	15	0
6	4	0	0	0	0	0	6	0	0	0	0	0	13	0	0	0	0	0	0	12	11	8	14	8	18	10	0	2	16	9	3	13	11	18	4	15	0
7	8	17	16	4	7	13	0	0	0	0	0	12	0	0	0	0	0	0	0	15	2	5	1	9	16	11	11	16	7	3	6	8	3	10	14	9	0
8	10	0	8	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	8	14	10	6	5	4	9	12	17	5	10	2	13	8	3	7	9	0
9	8	2	0	0	14	0	0	0	0	10	0	0	0	0	0	0	0	0	0	4	4	13	10	16	18	4	7	12	0	5	13	17	11	12	3	9	0
10	1	0	3	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	18	0	5	2	16	8	6	0	10	17	1	10	16	1	16	16	18	0
11	18	17	0	15	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	17	15	10	7	12	8	12	14	5	4	1	6	2	6	12	15	18	0
12	1	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	15	7	8	12	4	0	3	8	18	14	12	17	7	4	4	13	18	0
13	18	2	16	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	11	14	11	12	8	15	18	5	15	10	8	2	7	7	11	9	18	0
14	1	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2	17	0	6	3	5	15	6	16	15	0	14	13	18	1	18	0
15	18	17	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	2	15	3	7	18	15	3	13	7	11	1	17	11	18	4	18	0
16	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	2	18	10	16	17	13	4	9	9	4	13	17	16	10	18	0
17	18	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	13	5	0	17	18	3	10	114	11	11	14	1	10	3	18	0	
18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	1	11	2	12	3	13	4	14	5	15	6	16	7	17	8	18	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	