An Algebraic Framework for Silent Preprocessing with Trustless Setup and Active Security

Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl

Aarhus University

Abstract. Recently, number-theoretic assumptions including DDH, DCR and QR have been used to build powerful tools for secure computation, in the form of *homomorphic secret-sharing* (HSS), which leads to secure two-party computation protocols with succinct communication, and *pseudorandom correlation functions* (PCFs), which allow non-interactive generation of a large quantity of correlated randomness. In this work, we present a group-theoretic framework for these classes of constructions, which unifies their approach to computing distributed discrete logarithms in various groups. We cast existing constructions in our framework, and also present new constructions, including one based on class groups of imaginary quadratic fields. This leads to the first construction of two-party homomorphic secret sharing for branching programs from class group assumptions.

Using our framework, we also obtain pseudorandom correlation functions for generating oblivious transfer and vector-OLE correlations from number-theoretic assumptions. These have a *trustless, public-key setup* when instantiating our framework using class groups. Previously, such constructions either needed a trusted setup in the form of an RSA modulus with unknown factorisation, or relied on multi-key fully homomorphic encryption from the learning with errors assumption.

We also show how to upgrade our constructions to achieve active security using appropriate zero-knowledge proofs. In the random oracle model, this leads to a one-round, actively secure protocol for setting up the PCF, as well as a 3-round, actively secure HSS-based protocol for secure two-party computation of branching programs with succinct communication.

1 Introduction

Homomorphic secret sharing (HSS) [BGI16] can be seen as a relaxed form of fully-homomorphic encryption (FHE), where two non-colluding servers evaluate a function on private inputs without interaction. At the end of the computation, the servers each obtain a secret share, and these can be combined to obtain the result. At the core of existing HSS constructions is a procedure for *distributed discrete log*, where two parties are given group elements g_0, g_1 such that $g_1 = g_0 \cdot g^x$ for some fixed base g, and want to convert these multiplicative shares into additive shares x_0, x_1, where $x_1 = x_0 + x$ over the integers. The method from [BGI16], which is based on the decisional Diffie-Hellman (DDH) assumption, allows doing this conversion without interaction, however, there is an inherent correctness error. This results in significant extra work to ensure that the magnitude of the error is small. Moreover, the error cannot be made negligible. This limitation carries over to the final HSS construction, which has a non-negligible probability that the result of the computation is incorrect.

Recently, it was shown that the non-negligible correctness error of the DDH construction can be overcome, when switching to the Paillier [Pai99] or Damgård-Jurik [DJ01] cryptosystems based on the decisional composite residuosity (DCR) assumption. With these encryption schemes, which work over $\mathbb{Z}_{N^2}^*$ for an RSA modulus N, discrete logarithms can be computed in a distributed manner with a very simple and perfectly correct algorithm [OSY21, RS21]. This avoids the challenges of the DDH setting, by exploiting the fact that the messages in these schemes lie in a subgroup where solving discrete log is easy.

In [OSY21], the same distributed discrete log technique was used for several other applications in secure computation. In particular, they constructed *pseudorandom correlation functions* based on the Paillier and quadratic residuosity assumptions. A pseudorandom correlation function (PCF) is a way of generating two short, correlated keys, such that when evaluating the function on each of the keys, the two outputs are correlated in some secret manner. This generalizes the notion of a pseudorandom correlation generator [BCG+19], which only supports a bounded number of outputs. Examples of useful correlations for PCFs and PCGs are random oblivious transfer correlations, or secret-shared multiplication triples, which can be used in GMW-style multi-party computation protocols [GMW87] with very lightweight online computation.

An appealing feature of the PCFs from [OSY21] is that the PCF keys can be generated in a *public-key* manner, where after publishing just a single, short message, each party can locally derive their PCF key and compute the correlated randomness. However, a major drawback is that to achieve this public-key setup, the parties first need to have a trusted setup in the form of a public RSA modulus with unknown factorisation.

1.1 Our Contributions

It may seem from the previous work in [OSY21, RS21] that their efficient approach to distributed discrete log depends on very specific properties of Paillier, or more generally Damgård-Jurik encryption.

However, we show that this is not the case: in Section 3 we present a general framework, where we demonstrate that the approach from previous works can be phrased in terms of abstract group-theoretic properties. Naturally, the known methods based on Paillier and Damgård-Jurik become special cases of our framework, but we also show instantiations under different assumptions in Section 4.

Below, we describe the main applications of our framework to secure two-party computation, and the results obtained from our new instantiations.

Homomorphic secret sharing. We show in Section 5 that any instantiation of our framework that supports superpolynomially large plaintexts can be used to build homomorphic secret sharing for the class of polynomial size branching programs. This construction follows the same blueprint as previous works that obtain HSS for branching programs [BGI16, BKS19, OSY21, RS21]. Using this, two new instantiations of our framework imply two new constructions of HSS based on a flavour of the decisional Diffie-Hellman assumption for short exponents.

Firstly, we obtain HSS from a variant of the Joye-Libert cryptosystem [JL13, BHJL17], modified to work over a modulus that is a product of many small, distinct primes; compared with the analogous constructions based on Paillier, this has the advantage that ciphertexts are only a single element of $\mathbb{Z}_N$, and we can be more flexible in our choice of plaintext space, which is limited to $\mathbb{Z}_{N^s}$ otherwise. For a plaintext space modulo Q, we need to choose N such that $p-1, q-1$ are divisible by Q, so when Q is large we should clearly increase p, q to compensate, however, for reasonable sizes of Q the resulting ciphertext size should still be smaller than Paillier, which is an element of $\mathbb{Z}_{N^2}$.

Secondly, we obtain HSS from the DDH assumption in class groups of imaginary quadratic fields, based on the CL cryptosystem [CL15]. Class groups have recently seen many cryptographic applications, since they offer a way to generate a group of unknown order, without relying on any trusted setup to create the group parameters. Using class groups in HSS, we avoid the need for a setup with an RSA modulus where no party knows the factorization, instead only relying on a CRS that can be sampled with public randomness. For security, we rely on the DDH assumption with short exponents, where the short exponents are used to ensure that the secret key fits in the message space of the scheme, which allows us to easily encrypt functions of the secret key without introducing a circular security assumption.

Public-key pseudorandom correlation functions with trustless setup. Our starting point here is the PCFs from Paillier and quadratic residuosity from [OSY21], which give PCFs for generating vector-OLE and OT correlations, respectively.

PCFs, by definition, involve a setup procedure where a trusted dealer distributes a pair of short keys to the two parties. In [OSY21], it was shown that given a 1-round protocol for vector-OLE, where each party sends one parallel message, the PCF setup procedure can be replaced with a simple *public-key setup*, where each party publishes one message, which is then used to derive a PCF key. To realize the 1-round vector-OLE protocol, they give a dedicated construction based on distributed discrete log from Paillier, however, this still relies on a trusted setup in the form of an RSA modulus with unknown factorisation. We show in Section 6 that this construction can be generalised to work under any instantiation of our framework; with our class groups instantiation, we then obtain vector-OLE with a trustless setup. Put together with the PCFs from [OSY21], this leads to a public-key PCF with trustless setup for vector-

OLE based on the combination of class group assumptions and DCR, or one for OT by combining class groups and quadratic residuosity.

Active security. Given a public-key PCF, where after exchanging public keys, two parties can compute as much correlated randomness as they need, it is natural to ask, can this type of protocol be made actively secure? Although there are many ways of generically compiling passively secure protocols into active ones [GMW86, IPS08], we want to achieve something reasonably practical, in particular, to avoid using generic zero-knowledge techniques that require expressing group operations as circuits or similar. We show in Section 7 how to upgrade our PCFs to achieve active security, while preserving their public-nature by using Fiat-Shamir based NIZKs in the random oracle model. We do this via a careful combination of sigma protocols, which all make black-box use of the group, so avoid the complications of generic techniques. One challenge is that to build the public-key PCF, we need one party to prove that their input to the vector-OLE protocol corresponds to a secret key for an RSA modulus used in the PCF. As an essential tool, we use an integer commitment scheme which we show can be built from class groups and a trustless set-up. Thus, even our actively secure PCF does not need a trusted dealer. See the next section for details on the assumption required for this.

Finally, in the full version of the paper [ADOS22, Section 8], we also show how to add active security to our HSS construction. In the random oracle model, this gives a 3-round protocol for actively secure two-party computation of branching programs, which makes black-box use of the operations needed by our group-theoretic framework. Here, as well as proving that ciphertexts used to the secret-share HSS inputs are well-formed, we also need range proofs to ensure that the inputs are bounded in size.

A comparison to [OSY21] and [RS21]. As summarised above, previous work focuses its analysis on Paillier and Goldwasser-Micali [OSY21], and Damgård-Jurik [RS21]. Both [OSY21] and [RS21] describe how to solve distributed discrete log in the setting they study and use the techniques to build HSS for branching programs. In [OSY21], the authors also explain how to build public-key PCFs for OT and VOLE using the distributed discrete log techniques. All the constructions presented in [OSY21] and [RS21] rely on a trusted setup for the generation of a public RSA modulo of unknown factorisation.

The main contribution of this work is to generalise the techniques of [OSY21] and [RS21] to an abstract algebraic framework. We characterise the assumptions that the framework needs to satisfy to solve distributed discrete log, build HSS for branching programs and public-key PCFs for OT and VOLE. We present also new instantiations of the framework in addition to Paillier, Goldwasser-Micali and Damgård-Jurik, namely variants of the Joye-Libert cryptosystem and class groups. The latter allows us to build HSS and public-key PCFs that do not need trusted setups. Finally, while [OSY21] and [RS21] limit their study to passive security only, this work explains how to upgrade the constructions to active security obtaining implementable solutions that make black-box use of the underlying group.

1.2 An Overview of the Framework

In a nutshell, our framework consists of a large, finite group G, where $G = F \times H$. In the subgroup F, which is cyclic with generator f, discrete log is easy, and the order of F is public (whereas this is not the case for H). In the distributed discrete logarithm problem, two parties are given group elements $g_0, g_1 \in G$, with the condition that $g_0/g_1 = f^m$ for some message m. The goal is for the parties to convert this into shares m_0, m_1, where $m_0 + m_1 = m$ modulo the order of F. The crucial ingredient we need for distributed discrete log is a function we call a *coset labelling function*, which, for each coset C of F in G, maps all elements in C to a specific element in C. Existence of a coset labelling function turns out to be enough to solve distributed discrete log assuming that the two parties start from elements in the same coset, and it further turns out that this is sufficient to implement all our constructions, as long as some appropriate computational assumptions hold in G.

Instantiations. This framework easily encompasses previous constructions where distributed discrete logs are computed with Paillier, Damgård-Jurik, or Goldwasser-Micali ciphertexts. We also show that a natural variant of the Joye-Libert cryptosystem can be used (although it remains open to find a coset labelling function for the original Joye-Libert scheme, with plaintexts modulo 2^k). Finally, we give an instantiation based on class groups over imaginary quadratic fields. Here, we essentially apply the framework of the CL cryptosystem [CL15] for linearly homomorphic encryption, and combine it with the observation that the coset labelling function can be obtained via a special surjective map, which was previously used in the NICE cryptosystem [PT00] and its cryptanalysis [CJLN09].

Trustless Setup and the DXDH Assumption. For all applications of our framework, we rely on the standard DDH assumption in the group G. In settings where we need a trustless setup, we sometimes use a new assumption we call the *decisional cross-group Diffie-Hellman*, or DXDH, assumption. This states that for group elements $g, h \leftarrow G$ sampled with random coins ρ_g, ρ_h, and random exponents r, s,

$$(\rho_g, \rho_h, g, h, g^r, h^r) \cong (\rho_g, \rho_h, g, h, g^r, g^s)$$

This assumption arises in settings where we have a CRS with two group elements g, h, and we want the CRS to be public-coin. Having a public-coin CRS implies a trustless setup, since in practice the parties can derive randomness using e.g. a random oracle, and use this to sample the group elements. Note that in a standard cyclic DDH group (such as with elliptic curves), DXDH and DDH are equivalent because g, h always generate the same group, and furthermore, given a group element g it is easy to find some random coins that 'explain' it.

With class groups, however, this is not the case, since we are not aware of any invertible sampling algorithm, nor any method for sampling g and h such that they lie in the same subgroup.

Thus, when aiming for a trustless setup, we need DXDH. An additional complication of this setting is that the assumption makes it harder to use a CRS in security proofs: there is no way to introduce a trapdoor in the CRS by picking $h = g^t$ in the simulation, as we do not know how to explain the random coins used to sample h (without leaking t).

We note that recently, [CKLR21] presented zero-knowledge proofs built using integer commitments from class groups, which require a CRS (g, h) and the assumption that (g, h) is indistinguishable from (g, g^s). Note that this assumption is incompatible with a trustless setup: if the CRS contains the random coins used to sample g and h, then the assumption doesn't hold as it is hard for the simulator to come up with the random coins needed to explain sampling $h = g^s$.[1] However, in Section 7 we show that the same commitment scheme *does* permit a trustless setup under the DXDH assumption, and we use this in our zero-knowledge proofs to obtain active security.

Recap of the framework. We now summarise the description of our framework. Our setting is an finite, Abelian group

$$G \cong F \times H \qquad \text{where} \qquad F = \langle f \rangle.$$

The group G needs to satisfy these properties:

1. The discrete log function over F is efficiently computable.
2. There exists an efficiently computable coset labelling function π.
3. There exists an efficiently computable function δ (the lifting function) such that $\pi\big(\delta(x)\big) = x$ for every input x.

In order to build HSS for branching programs and public-key PCFs for OT and VOLE, the group needs to satisfy additional computational assumption which are summarised in Table 1.

Construction	***Assumptions and Model***
HSS for branching programs	DDH, small exponent (DXDH, weak hidden order + RO)
Public-Key PCF for VOLE	DCR, DXDH, DDH (weak hidden order, QR) + RO
Public-Key PCF for OT	QR, DXDH, DDH (weak hidden order) + RO

Table 1. Computational assumptions needed by our constructions. Elements written in between brackets are needed only for active security.

[1] The authors of [CKLR21] have acknowledged. They claim to have found a solution and are going to update their work.

2 Notation and Preliminaries

Let λ denote the security parameter. Our constructions are restricted to the two-party setting and we denote them P_0 and P_1. For any $a, b \in \mathbb{Z}$ with $a < b$, we represent the set of integers $\{a, a+1, \dots, b\}$ by $[a, b]$. We use $[b]$ to represent $[0, b-1]$. We assume that by reducing an element modulo $t \in \mathbb{N}$, we obtain a value in $[t]$.

Given a deterministic algorithm Alg, we denote its evaluation on an input x and the assignment of the result to a variable y by $y \leftarrow \mathsf{Alg}(x)$. If Alg is instead probabilistic, we write $y \overset{R}{\leftarrow} \mathsf{Alg}(x)$. The operation assumes that the random bits used by the algorithm are sampled uniformly. When we want to use a specific random string r, we write instead $y \leftarrow \mathsf{Alg}(x; r)$. Finally, if the element y is uniformly sampled from a set $\mathcal{X}$, we write $y \overset{R}{\leftarrow} \mathcal{X}$.

We denote vectorial elements using the bold font, the i-th entry of a vector $\boldsymbol{v}$ is denoted by v_i or by $\boldsymbol{v}[i]$. The cyclic subgroup generated by a group element g is represented by $\langle g \rangle$. Finally, we denote secret-shared elements y using the y-in-a-box notation, i.e. $[y]$. It will be clear from the context if that denotes a secret-sharing or the set $\{0, 1, \dots, y-1\}$.

In the full version of the paper [ADOS22, Section 2], we provide an overview on homomorphic secret-sharing (HSS) and pseudorandom correlation functions (PCFs).

3 A Group-Theoretic Framework

We will assume we have a probabilistic polynomial time algorithm Gen that takes $\mathbb{1}^\lambda$ as input where λ is a security parameter. When running Gen, we get output

$$\mathsf{par} \overset{R}{\leftarrow} \mathsf{Gen}(\mathbb{1}^\lambda), \text{ where } \mathsf{par} = (G, F, H, f, t, \ell, \mathsf{aux}).$$

Here, G is a finite Abelian group with subgroups F, H such that $G = F \times H$, f is a generator of F and t is the order of F. We assume we can compute the group operation and inverses in time polynomial in λ. The natural number ℓ will be used in the following: when we select a random exponent r and compute g^r where $g \in G$, r will usually be chosen uniformly between 0 and ℓ[2]. Finally, we say that Gen is *public-coin* if the random coins used by Gen appear in the string aux.

We also assume a probabilistic polynomial time algorithm $\mathcal{D}$ for sampling random elements in G. We will use the notation $(g, \rho) \overset{R}{\leftarrow} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$, where $g \in G$ is the sampled element and ρ contains the random coins used in the sampling (i.e., the sampling of g is always *public-coin*). We do not require that g is uniform in G, but we do require f is in the subgroup generated by g, except perhaps with negligible probability.

[2] We will always choose ℓ large enough so that g^r is statistically indistinguishable from uniform in $\langle g \rangle$. This is possible, even if $|H|$ is sometimes not known by anyone, since an upper bound is always known.

We assume that discrete log base f is easy, that is, given f^a for any $a \in \mathbb{Z}_t$, a can be computed in polynomial time in λ.

In the following sections, we will specify a number of computational problems that we need to assume are hard to solve, given par and various elements sampled by $\mathcal{D}$. Loosely speaking, the most basic one is that the order of the subgroup H is hard to compute, and that the DDH assumption holds in the subgroup generated by g where g is sampled by $\mathcal{D}$. More details will be given in Section 3.1.

The main problem we want to solve in the context of the framework is the following, which we call Non-Interactive Discrete Log Sharing (NIDLS). This is defined as follows:

Definition 1. *The NIDLS problem involves two parties, A and B. A gets as input $\alpha \in G$, while B gets $\beta \in G$. It is promised that $\alpha\beta^{-1} \in F$, so that $\alpha\beta^{-1} = f^m$ for some $m \in \mathbb{Z}_t$. A and B now do only local computation and A outputs a number a, while B outputs b. The goal is that $a + b \equiv m \bmod t$.*

It will be convenient to introduce the following notation: for $g \in G$, we denote by C_g be the coset of F in G that contains g. As we explain in a moment, the NIDLS problem can be solved using the following tool:

Definition 2. *A* coset labelling function for F in G *is an efficiently computable function $\phi : G \mapsto G$ with the following property: for any $g \in G$ we have $\phi(g) \in C_g$ and furthermore, for any $h \in C_g$ we have $\phi(h) = \phi(g)$.*

In other words, for every coset C_g, ϕ defines a fixed element $c \in C_g$ and c can be efficiently computed given any element in C_g.

Given a coset labelling function the NIDLS problem can be solved using the following protocol:

1. A computes $\phi(\alpha)^{-1} \cdot \alpha$ which is in F since α and $\phi(\alpha)$ are in the same coset. Using that discrete log in F is easy, A computes a such that $\phi(\alpha)^{-1} \cdot \alpha = f^a$, and outputs a.
2. B computes $\phi(\beta) \cdot \beta^{-1}$ which is in F since β and $\phi(\beta)$ are in the same coset. Using that discrete log in F is easy, B computes b such that $\phi(\beta) \cdot \beta^{-1} = f^b$, and outputs b.

This works because the property of ϕ guarantees that $\phi(\alpha) = \phi(\beta)$. Therefore

$$f^a \cdot f^b = \phi(\alpha)^{-1} \cdot \alpha \cdot \phi(\beta) \cdot \beta^{-1} = \alpha \cdot \beta^{-1} = f^m,$$

from which it follows immediately that $a + b \equiv m \bmod t$.

It turns out that if F is small, then a coset labelling function always exists:

Lemma 1. *Let $G = F \times H$ be groups as described above, where the order t of F is polynomial. Then a coset labelling function for F in G always exists.*

Proof. We define the desired function ϕ as follows: on input g, compute a list of all elements in C_g by multiplying g by all powers of f. This is feasible since t is polynomial. Sort the elements in lexicographical order and output the first element. As the content of the list is the same no matter which element in the coset we start from, this function has the desired property. $\square$

There is also a different approach to constructing a coset labelling function which, as we shall see, sometimes works for superpolynomial size F.

Namely, assume that for every G that Gen can produce, there exists an efficiently computable and surjective homomorphism $\pi : G \mapsto G'$ (for some group G'), where $ker(\pi) = F$. This implies that for each coset of F in G, π maps all elements of the coset to a single element in G', and that distinct cosets are mapped to distinct elements.

Note that $\pi(g)$ is actually a unique "label" for the coset C_g, the only problem is that it is in G' and not in G.

To get around this, we assume that outputs from π can be "lifted" deterministically to G such that we land in the coset we came from. That is, we assume there exists an efficiently computable function $\delta : G' \mapsto G$ such that for any $x \in G'$ we have that $\delta(x)$ is in the coset of F in G that is mapped to x by π. Put slightly differently, what we want is that $\pi(\delta(x)) = x$ for all $x \in G'$.

Now, observe that $\delta(\pi(g))$ only depends on which coset g belongs to, since $\pi(g)$ already has this property. Therefore, the following lemma is immediate:

Lemma 2. *Let $G = F \times H$, G' be groups as described above and π, δ be functions as described above, with $\pi(\delta(x)) = x$ for all $x \in G'$. Then ϕ defined by $\phi(g) = \delta(\pi(g))$ is a coset labelling function for F in G.*

3.1 Assumptions

In this section we list the computational assumptions we need in order to prove our constructions secure.

Definition 3 (Weak Hidden Order Assumption). *We say that the weak hidden order assumption holds in the NIDLS framework if for any PPT adversary $\mathcal{A}$:*

$$\Pr[\mathcal{A}(\mathsf{par}, g, \rho) = x \text{ and } g^x = 1] = \mathsf{negl}(\lambda)$$

when $\mathsf{par} := (G, F, H, f, t, \ell, \mathsf{aux}) \overset{R}{\leftarrow} \mathsf{Gen}(1\!\!1^\lambda)$ *and* $(g, \rho) \overset{R}{\leftarrow} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$.

Notice that in the standard hidden order assumption [Tuc20], the adversary is let free to choose any $g \neq 1$. We rely instead on a *weaker* assumption in which g is sampled according to $\mathcal{D}$.

Definition 4 (DDH Assumption). *We say that the DDH assumption holds in the NIDLS framework if for any PPT adversary $\mathcal{A}$ the following quantity is negligible:*

$$|\Pr[\mathcal{A}(\mathsf{par}, \rho, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(\mathsf{par}, \rho, g, g^x, g^y, g^z) = 1]| = \mathsf{negl}(\lambda)$$

when $\mathsf{par} := (G, F, H, f, t, \ell, \mathsf{aux}) \overset{R}{\leftarrow} \mathsf{Gen}(\mathbb{1}^\lambda), (g, \rho) \overset{R}{\leftarrow} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par}), (x, y, z) \overset{R}{\leftarrow} [\ell]^3$.

We introduce a new variant of the DDH assumption that allows us to infer the security of our protocols that use two generators g, C which are generated with a trustless setup i.e., the adversary is allowed to see the random coins used for their generation. In some settings, this assumption is equivalent to DDH but this does not cover all our instantiations of the framework[3].

Definition 5 (Decisional Cross-Group DH Assumption (DXDH)). *We say that the DXDH assumption holds in the NIDLS framework if for any PPT adversary $\mathcal{A}$:*

$$|\Pr[\mathcal{A}(\mathsf{par}, g, \rho_0, C, \rho_1, g^r, C^r) = 1] - \Pr[\mathcal{A}(\mathsf{par}, g, \rho_0, C, \rho_1, C^s, C^r) = 1]| = \mathsf{negl}(\lambda)$$

when $\mathsf{par} := (G, F, H, f, t, \ell, \mathsf{aux}) \overset{R}{\leftarrow} \mathsf{Gen}(\mathbb{1}^\lambda)$*,* $(g, \rho_0) \overset{R}{\leftarrow} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$*,* $(C, \rho_1) \overset{R}{\leftarrow} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$*,* $C \neq g$ *and* $(r, s) \overset{R}{\leftarrow} [\ell]^2$.

Finally, in our HSS constructions, we would like to have ElGamal-style secret keys bounded by $\ell_{\mathsf{sk}} < t$, which may be significantly smaller than ℓ. This allows to encrypt the private key under its public counterpart without worrying about wrap-arounds. In order for security to hold in these conditions, we rely on the small exponent assumption defined below.

Definition 6 (Small Exponent Assumption). *We say that the small-exponent assumption with length $\ell_{\mathsf{sk}}(\lambda)$ holds in the NIDLS framework if for any PPT adversary $\mathcal{A}$:*

$$|\Pr[\mathcal{A}(\mathsf{par}, \ell_{\mathsf{sk}}, g, \rho, g^x) = 1] - \Pr[\mathcal{A}(\mathsf{par}, \ell_{\mathsf{sk}}, g, \rho, g^y) = 1]| = \mathsf{negl}(\lambda)$$

when $\mathsf{par} := (G, F, H, f, t, \ell, \mathsf{aux}) \overset{R}{\leftarrow} \mathsf{Gen}(\mathbb{1}^\lambda)$*,* $(g, \rho) \overset{R}{\leftarrow} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$*,* $x \overset{R}{\leftarrow} [\ell]$ *and* $y \overset{R}{\leftarrow} [\ell_{\mathsf{sk}}]$.

4 Instantiations of the Framework

In this section, we give a number of concrete instantiations of the framework we just discussed. Some were already known, and some are new.

4.1 Paillier and Damgård-Jurik

This example was already known from [OSY21] who presented a NIDLS protocol based on Paillier encryption and independent work from [RS21] who did it from Damgård-Jurik encryption.

[3] For equivalence, it is needed that g and C are random generators of the same subgroup and that $\mathcal{D}$ is invertible, i.e., that given any group element h in the output domain, one can efficiently compute random coins that would cause $\mathcal{D}$ to output h.

These instantiations are closely related and we cover them in one go as follows: we let $\mathsf{Gen}(1\!\!1^\lambda)$ output an RSA modulus $n = pq$ of bit length λ, where $p' = (p-1)/2$ and $q' = (q-1)/2$ are also prime and where $gcd(n, \phi(n)) = 1$. We set $G = \mathbb{Z}_{n^s}^*$, for some constant natural number $s \geq 2$ and it now holds that $G = F \times H$ where F is the subgroup of order n^{s-1}, and H is the subgroup of order $(p-1)(q-1)$. Discrete log in F is easy in this case (see [DJ01] for details). This generator is not public-coin, as the prime factors of n must remain secret.

To get a coset labelling function for this example, we use Lemma 2: we set $G' = \mathbb{Z}_n^*$ and $\pi(g) = g \bmod n$. Since n divides n^s, it is clear that π is a surjective homomorphism from G to G'. Therefore its kernel has order $|G|/|G'| = n^{s-1}$. Note that all non-trivial elements in F must have orders relatively prime to $\phi(n) = |G'|$ and hence the homomorphism into G' must send all these elements to 1. It follows that F is contained in the kernel and so is in fact equal to the kernel because $|F| = n^{s-1}$. We define the function $\delta : G' \mapsto G$ by $\delta(x) = x$, that is, δ just returns its input, but now understood as a number modulo n^s (instead of n).

With these definitions, it is clear that $\pi(\delta(x)) = x$, so by Lemma 2, $\phi(g) = \delta(\pi(g))$ is a coset labelling function.

The sampling algorithm $\mathcal{D}$ will output a random $g \in \mathbb{Z}_{n^s}^*$, such that the Jacobi symbol of g modulo n is 1. Note that because $|F| = n^{s-1}$ contains only large prime factors, a random g will contain F in the subgroup it generates except with negligible probability. Similarly, reducing modulo n, we see that $g \bmod n$ has order divisible by $p'q'$ except with negligible probability since p', q' are prime.

As for the assumptions, computing the order is trivially equivalent to factoring n. The DDH assumption was introduced in [DJ03] and used there for an "El-Gamal style" variant of Paillier encryption. In this setting, we can claim that if you can break the DXDH assumption, you can also break DDH. This is because g (or C) sampled as above have order $n^{s-1}p'q'$ or $2n^{s-1}p'q'$ except with negligible probability. Whether 2 divides the order cannot be efficiently determined (by the standard quadratic residuosity assumption). Further, the sampling algorithm is clearly invertible. All this means that, given an element g^x from a DDH challenge, we can claim it was instead sampled by $\mathcal{D}$ and let it play the role of C in the DXDH setting.

Finally, the small exponent assumption is reasonable in a setting where discrete log and DDH are hard, as we do assume here, as long as the domain from which the exponent is chosen is exponentially large. Also, this type of assumption has been used several times before, for instance in [BCG+17] to optimize an HSS construction.

4.2 Joye-Libert Variants

Small order *F*. In this example, the generator outputs an RSA modulus $n = pq$ where 2^ℓ is the maximal 2-power that divides $p-1$, and $q-1$. It also outputs an element $f \in \mathbb{Z}_n^*$ of order 2^ℓ modulo both p and q (and so it also has order 2^ℓ modulo n). Let $p' = (p-1)/2^\ell, q' = (q-1)/2^\ell$, where we assume that p', q'

are prime. Then we let $F = \langle f \rangle$, we let $H \leq \mathbb{Z}_n^*$ be the subgroup of order $p'q'$, and we set $G = F \times H$. The group G is actually not all of $\mathbb{Z}_n^*$, but this is of no consequence in the following. Discrete log in F is easy by the Pohlig-Hellman algorithm.

For this variant, as long as 2^ℓ is polynomial, we can use Lemma 1 to get a coset labelling function. Doing it for larger values of 2^ℓ is an open problem. When $\ell = 1$, we can set $f = -1$ and we get a setting closely related to the Goldwasser-Micali cryptosystem, as observed in [OSY21].

Large order F. We now construct a different variant of the Joye-Libert case where we are able to accommodate an exponentially large order subgroup F. Once again, the generator outputs an RSA modulus $n = pq$. This time, both $p-1$ and $q-1$ are divisible by the product of the first ℓ primes q_ℓ, that is $q_\ell = \prod_{i=1}^{\ell} p_i$ where p_i is the i'th prime.

We let $f \in \mathbb{Z}_n^*$ be an element of order q_ℓ modulo both p and q. Let $p' = (p-1)/q_\ell, q' = (q-1)/q_\ell$. As before, we let $F = \langle f \rangle$, we let $H \leq \mathbb{Z}_n^*$ be the subgroup of order $p'q'$, and we set $G = F \times H$.

It is not hard to see that since the i'th prime is approximately $i \ln i$, we can arrange for q_ℓ to be exponentially large, while each prime in the product is only polynomial.

We now show that if all primes in the product q_ℓ are polynomial size, we can solve the NIDLS problem in this setting, basically by using Lemma 1 for each p_i and then assembling a complete solution using the Chinese remainder theorem (CRT).

Some notation: we have $F = F_1 \times ... \times F_\ell$, where F_i is of order p_i. So it follows from Lemma 1 that we have a coset labelling function ϕ_i for the group $G_i = F_i \times H$. Also, if we let $u_i = q_\ell / p_i$, then $f_i = f^{u_i}$ is a generator of F_i. Now observe that if α, β is an instance of the NIDLS problem in $G = F \times H$, then $\alpha^{u_i}, \beta^{u_i}$ is an instance of the NIDLS problem in $G_i = F_i \times H$. This is simply because $\alpha \cdot \beta^{-1} = f^m$ implies $\alpha^{u_i} \cdot (\beta^{u_i})^{-1} = (f^{u_i})^m = f_i^{m \bmod p_i}$. Using this notation, the protocol works as follows:

1. For each $i = 1...\ell$, A uses ϕ_i to compute a solution a_i to the NIDLS problem in G_i. Finally, using CRT, A computes and outputs $a \in \mathbb{Z}_t$ such that $a \bmod p_i = a_i$ for all i.
2. For each $i = 1...\ell$, B uses ϕ_i to compute a solution b_i to the NIDLS problem in G_i. Finally, using CRT, B computes and outputs $b \in \mathbb{Z}_t$ such that $b \bmod p_i = b_i$ for all i.

This works because $(a+b) \bmod p_i = (a_i + b_i) \bmod p_i$ by definition of a, b, and since a_i, b_i solves the NIDLS problem in G_i we further have

$$(a+b) \bmod p_i = (a_i + b_i) \bmod p_i = m \bmod p_i.$$

Since this holds for all i, CRT implies that $a + b \bmod q_\ell = m$.

For this instantiation, the sampling algorithm $\mathcal{D}$ will choose a random $r \in \mathbb{Z}_n^*$ and output $g = f \cdot r^{q_\ell} \bmod n$. Note that $r^{q_\ell} \bmod n$ has order $p'q'$ except with negligible probability, in particular, the order is prime to q_ℓ so g has order $q_\ell p'q'$, and hence f is in the group generated by g.

The assumptions for this instantiation can be motivated similarly to what was done for Paillier above, as also here we rely on factoring to hide the order of the group. For this to be reasonable, we need, of course, that q_ℓ is much smaller than n so that enough uncertainty remains about p, q even given q_ℓ. The exception is that in this case, $\mathcal{D}$ is not invertible, so we cannot claim that DDH implies DXDH. The assumptions are also closely related to what Joye and Libert [JL13] assumed for their cryptosystem, but one should note that our assumptions are stronger because we need to make an element of order exactly 2^ℓ (or q_ℓ) public, while they just needed an element of order divisible by 2^ℓ. When 2^ℓ is small, such an element can be guessed with good probability while it is not clear how to efficiently compute an element of order exactly 2^ℓ given only n.

4.3 Class Groups

We explain here how to instantiate our framework on top of the CL framework [CL15] (see also [Tuc20] for an excellent introduction to class groups). Basically, we take the CL framework, and combine this with the observation that a coset-labelling function can be obtained from a surjective homomorphism used previously in the NICE cryptosystem [PT00, CJLN09].

Let $\mathsf{Gen}(1\!\!1^\lambda)$ output two primes p and q such that $pq \equiv 3 \pmod 4$ and $(p/q) = -1$. This generator is public-coin, p and q will be public. We set $\Delta_K = -pq$ and $\Delta_q = -pq^3$. We set $G = Cl(\Delta_q)$, the class group of the quadratic order $\mathcal{O}_{\Delta_q}$ of discriminant Δ_q and $G' = Cl(\Delta_K)$ the class group of the maximal order $\mathcal{O}_{\Delta_K}$. The size of pq is chosen such that computing the class number $|G'|$ is intractable.

Let $f \in G$ be the class of the ideal $q^2\mathbb{Z} + (-q + \sqrt{\Delta_q})/2\mathbb{Z}$ then f has order q and the discrete logarithm problem in F, generated by f, is easy.

If q has λ bits then q is prime to $|G'|$ except with negligible probability by the Cohen-Lenstra heuristics. Then $G \simeq F \times H$ where H is a subgroup of order $|G'|$.

We denote by $I(\mathcal{O}_{\Delta_q}, q)$ (resp. $I(\mathcal{O}_{\Delta_K}, q)$) the subgroup of fractional ideals generated by $\mathcal{O}_{\Delta_q}$-ideals prime to q (resp. of $\mathcal{O}_{\Delta_K}$-ideals prime to q). Then, the map $\varphi_q : I(\mathcal{O}_{\Delta_q}, q) \to I(\mathcal{O}_{\Delta_K}, q)$, $\mathfrak{a} \mapsto \mathfrak{a}\mathcal{O}_{\Delta_K}$ is an isomorphism. The reverse map is $\varphi_q^{-1} : I(\mathcal{O}_{\Delta_K}, q) \to I(\mathcal{O}_{\Delta_q}, q)$, $\mathfrak{a} \mapsto \mathfrak{a} \cap \mathcal{O}_{\Delta_q}$. Both maps are efficiently computable knowing q. The map φ_q induces a surjective homomorphism from G to G'. This will be the surjection π of the framework. The kernel of π is F.

We then define the function $\delta : G' \mapsto G$ by $\delta(x) = [\varphi_q^{-1}(\mathfrak{a})]$ where $\mathfrak{a}$ is an ideal in the class of x prime to q (it can also be found efficiently).

We then have $\pi(\delta(x)) = x$ by construction, so by Lemma 2, $\phi(g) = \delta(\pi(g))$ is a coset labelling function.

As sampling algorithm $\mathcal{D}$ we use the one introduced in [CL15], and also described in [Tuc20], section 3.1.2. It outputs g of large order such that f is guaranteed to be in the subgroup generated by g. Very briefly, it works by selecting a small prime r such that Δ_K is a square modulo r. From this r, we can construct an element in G' by considering the ideal that lies “above r” and the class of this ideal squared. We then lift this element to G as explained above, to get a group element h. Finally, we output $g = f \cdot h^t$.

With this sampling algorithm, the DDH assumption is the same that has been used before in the CL framework, sometimes known as the DDH-CL assumption. The DXDH assumption in this setting is not implied by DDH, since elements sampled from different randomness do not necessarily generate the same group. Nevertheless, we can argue that the assumption is reasonable: to break it, one needs to decide, for given g, C if a pair of group elements is of form g^r, C^r. The natural approach to this is to use index calculus type methods to find a relation of form $g^a = C^b$ which, for a pair of the form mentioned would imply $(g^r)^a = (C^r)^b$. However, once such an attack succeeds one would also be in a position to find orders of elements and hence break the (much more standard) hidden order assumption.

5 HSS Constructions

In this section, we explain how any instantiation of the framework can be used to build a cryptosystem and a homomorphic secret-sharing scheme (HSS) for restricted multiplication straight-line programs (RMS). Note that given the NIDLS-ElGamal encryption and a distributed DDLOG procedure, constructing an HSS follows in a more or less direct way by following the blueprint of the HSS in [OSY21]. However, since upcoming sections build on top of the HSS we provide the full description of the HSS anyway to make the paper self-contained.

5.1 NIDLS ElGamal

Our HSS construction is based on an ElGamal-style encryption scheme instantiated over our group-theoretic framework. We refer to the construction by *NIDLS ElGamal*, the cryptosystem is formally described in Fig. 1. Correctness of the construction follows immediately as for standard ElGamal.

CPA Security. Similarly to [CL15], the security of NIDLS ElGamal is implied by the DDH assumption, which states that random tuples (g, g^x, g^y, g^{xy}) are indistinguishable from (g, g^x, g^y, g^z). Since $\mathcal{D}$ outputs elements g for which $f \in \langle g \rangle$, we can use g^z to hide f^x.

Generating encryptions of the secret key. Note that in addition to the standard algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, we have included an additional algorithm SkEnc which encrypts the message “in the wrong place”. It turns out that this results in a valid encryption of the value $s \cdot x \bmod t$ i.e., an encryption of the secret key s times the input value x. In particular

$$c_1 \cdot c_0^{-s} = h^r \cdot (g^r \cdot f^{-x})^{-s} = (g^{rs} \cdot g^{-rs}) \cdot f^{sx}$$

This will be useful in our HSS construction. Formal proofs of the security of the scheme are in the full version of the paper [ADOS22, Section 5.1].

ElGamal Cryptosystem

$\mathsf{EG.Gen}(1^\lambda)$:

1. Sample $\mathsf{par} := (G, F, H, f, t, \ell, \mathsf{aux}) \overset{R}{\leftarrow} \mathsf{Gen}(\mathbb{1}^\lambda)$
2. Sample a random $(g, \rho) \overset{R}{\leftarrow} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$
3. Sample a random $s \overset{R}{\leftarrow} [\ell]$, and let $h = g^s$
4. Output $\mathsf{pk} = (\mathsf{par}, g, \rho, h)$ and $\mathsf{sk} = s$.

$\mathsf{EG.Enc}(\mathsf{pk}, x)$:

1. Sample a random $r \overset{R}{\leftarrow} [\ell]$
2. Output $\mathsf{ct} = (g^r, h^r \cdot f^x)$

$\mathsf{EG.Dec}(\mathsf{sk}, \mathsf{ct} = (c_0, c_1))$:

1. Output $x = \mathsf{DLog}_f(c_1 \cdot c_0^{-s})$

$\mathsf{EG.SkEnc}(\mathsf{pk}, x)$:

1. Sample a random $r \overset{R}{\leftarrow} [\ell]$
2. Output $\mathsf{ct} = (g^r \cdot f^{-x}, h^r)$

Fig. 1. A description of the ElGamal cryptosystem in the NIDLS framework.

5.2 Public-Key HSS

We now present a homomorphic secret-sharing scheme (HSS) for RMS programs based on the NIDLS framework. The main advantage of our NIDLS-based HSS compared to the Paillier-based HSS of [OSY21] is that we remove any need for trusted setups when instantiating the NIDLS over class groups, while previous constructions had to rely on a trusted dealer for the generation of an RSA modulus.

RMS programs. Restricted multiplications straight-line (RMS) programs are arithmetic circuits over $\mathbb{Z}$ that never compute multiplications between two intermediate value of the computation: at least one of the two factors must be an input. Intermediate values of the computation are often referred to as memory values. This class includes also branching programs, which likewise contains NC^1.

Definition 7 (RMS Programs). *An RMS program consists of a bound $B \in \mathbb{N}$, a modulo $n_{\mathsf{out}} \in \mathbb{N}$ and a polynomial-sized circuit in which the only gate types allowed are the following.*

– $\mathsf{ConvertInput}(\mathsf{I}_x) \to \mathsf{M}_x$. *Load the value of the input wire I_x to the memory wire M_x.*

– $\mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y) \rightarrow \mathsf{M}_z$. *Add the values of the memory wires* M_x *and* M_y *and assign the result to the memory wire* M_z.
– $\mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y) \rightarrow \mathsf{M}_z$. *Multiply the value of the input wire* I_x *by the value of the memory wire* M_y. *Assign the result to the memory wire* M_z.
– $\mathsf{Output}(\mathsf{M}_z) \rightarrow z$. *Output the value of the memory wire* M_z *reducing it modulo* n_{out}.

The circuit accepts only integral inputs. Whenever the absolute value $|x|$ *of any wire exceeds the bound* B*, the output of the execution is* $\perp$.

The public-key HSS scheme. We are now ready to present our construction, which is formally described in Fig. 2. We discuss the main ideas.

Our HSS scheme allows two parties to non-interactively apply an RMS program C on secret-shared inputs, obtaining additively secret-shared outputs. The scheme relies on a setup procedure[4] that provides the parties with a PRF key k, a NIDLS ElGamal public key pk, and a subtractive secret-sharing over the integers of the private counterpart $s = s_1 - s_0$. We assume that the length $\mathsf{len}_{\mathsf{sk}}$ of the private key is sufficiently small, so that $s < t$. If this condition is not satisfied, we need to proceed as in [OSY21], splitting the private key into small blocks and providing the parties with an encryption of each of them.

Input wires and memory wires. During the evaluation of the circuit C, each input wire I_x is associated with two NIDLS ElGamal ciphertexts: an encryption of the value of the wire x and an encryption of the product between x and the ElGamal secret key s. Such ciphertexts are produced and broadcast by the party providing the input. Remember that one does not need to know s in order to encrypt $x \cdot s$. Indeed, the algorithm SkEnc described in Section 5.1 can be used instead. Each memory wire M_x is instead associated with two subtractive secret-sharings over the integers: a secret-sharing of the value of the wire x and a secret-sharing of $x' := x \cdot s$.

Linear operations. Performing additions between memory values is straightforward due to the linearity of subtractive secret-sharing, i.e. to add M_x and M_y, it is sufficient to compute $[z] \leftarrow [x]+[y]$ and $[z'] \leftarrow [x']+[y'] = [x\cdot s]+[y\cdot s]$. Observe that additions allow us to model also multiplications by public constants in $\mathbb{Z}$.

Multiplications between input wires and memory wires. Multiplications between input wires and memory wires require more interesting techniques based on DDLOG. Let $\mathsf{ct}_x = (c_0, c_1)$ be the ElGamal encryption of x, the value of the input wire I_x. Moreover, let $[y]$ and $[y' = y \cdot s]$ be the subtractive secret-sharings associated with the memory wire M_y. In particular, the parties P_0 and P_1 own integers y_0, y'_0 and y_1, y'_1 such that $y_1 = y_0 + y$ and $y'_1 = y'_0 + y \cdot s$. Now, observe that $c_1^{y_0} \cdot c_0^{-y'_0}$ and $c_1^{y_1} \cdot c_0^{-y'_1}$ are a divisive secret-sharing of f^{xy}. Indeed,

$$c_1^{y_1} \cdot c_0^{-y'_1} = c_1^{y_0+y} \cdot c_0^{-(y'_0+y\cdot s)} = (c_1 \cdot c_0^{-s})^y \cdot c_1^{y_0} \cdot c_0^{-y'_0} = f^{xy} \cdot c_1^{y_0} \cdot c_0^{-y'_0}.$$

[4] Following the blueprint of [OSY21], it is possible to substitute the setup with a one-round protocol.

```
HSS Scheme

Setup($\mathbb{1}^\lambda$):
    1. Let $\mathsf{par} := (G, F, H, f, t, \ell, \mathsf{aux}) \xleftarrow{R} \mathsf{Gen}(\mathbb{1}^\lambda)$ and $\ell_{\mathsf{sk}}$ be the parameter for
       the small-exponent assumption.
    2. $(g, \rho) \xleftarrow{R} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$
    3. $s_0, s_1 \xleftarrow{R} [\ell_{\mathsf{sk}}]$
    4. $\mathsf{pk} \leftarrow g^{s_1} \cdot g^{-s_0}$
    5. $k \xleftarrow{R} \{0,1\}^\lambda$
    6. Output $\big(\mathsf{par}, g, \rho, \ell_{\mathsf{sk}}, \mathsf{pk}, k, (s_0, s_1)\big)$.
Input($\mathsf{pk}, x$):
    1. $\mathsf{ct}_x \xleftarrow{R} \mathsf{EG.Enc}(\mathsf{pk}, x)$
    2. $\mathsf{ct}_{xs} \xleftarrow{R} \mathsf{EG.SkEnc}(\mathsf{pk}, x)$
    3. Output $\mathsf{I}_x \leftarrow (\mathsf{ct}_x, \mathsf{ct}_{xs})$.
Eval$\big(i, s_i, (\mathsf{I}^1, \mathsf{I}^2, \ldots, \mathsf{I}^n), P\big)$:
  Party $P_i$ evaluates the RMS program $P$ gate by gate as follows.
    – $\mathsf{M}_x \leftarrow \mathsf{ConvertInput}(\mathsf{I}_x)$:
      Compute $\mathsf{M}_x \leftarrow \mathsf{Mult}\big(\mathsf{I}_x, \mathsf{M}_1 := (i, s_i)\big)$.
    – $\mathsf{M}_z \leftarrow \mathsf{Add}(\mathsf{M}_x, \mathsf{M}_y)$:
      Compute $z_i \leftarrow x_i + y_i$ and $z'_i \leftarrow x'_i + y'_i$ and set $\mathsf{M}_z \leftarrow (z_i, z'_i)$.
    – $\mathsf{M}_z \leftarrow \mathsf{Mult}(\mathsf{I}_x, \mathsf{M}_y)$:
      Let $\mathsf{ct}_x = (c_0, c_1)$ and $\mathsf{ct}_{xs} = (d_0, d_1)$. Let $\mathsf{id}$ be the label of the gate.
        1. $z_i \leftarrow (-1)^{1-i} \cdot \mathsf{DDLog}\big(c_1^{y_i} \cdot c_0^{-y'_i}\big) + \mathsf{F}_k(\mathsf{id}, 0) \bmod t$
        2. $z'_i \leftarrow (-1)^{1-i} \cdot \mathsf{DDLog}\big(d_1^{y_i} \cdot d_0^{-y'_i}\big) + \mathsf{F}_k(\mathsf{id}, 1) \bmod t$
        3. $\mathsf{M}_z \leftarrow (z_i, z'_i)$
    – $\mathsf{Output}(\mathsf{M}_z)$:
        1. Output $(-1)^{1-i} \cdot z_i \bmod n_{\mathsf{out}}$
```

Fig. 2. The HSS scheme for RMS programs based on the NIDLS framework.

By applying DDLOG on the respective divisive shares, the parties are therefore able to obtain a secret-sharing of the product $x \cdot y$ over $\mathbb{Z}_t$ (we recall that $t := \mathsf{ord}(f)$). By repeating the procedure for the other ciphertext associated with the input wire I_x, namely the encryption of $x \cdot s$, the parties can non-interactively obtain also a secret-sharing of $x \cdot y \cdot s$. Observe that the additive secret-sharings over $\mathbb{Z}_t$ can be easily converted into subtractive ones by simply changing the signs of the shares of P_0. In order to be sure that the shares are random over $\mathbb{Z}_t$, we rerandomise them using the PRF key k. As a consequence, as long as $|x \cdot y \cdot s| \ll t$, with overwhelming probability, the difference of the shares does not wrap around t, so the parties actually obtain a subtractive secret-sharing over $\mathbb{Z}$.

Input conversions and outputs. It remains to explain how to perform the input conversions and how to retrieve the outputs. Both operations are now rather straightforward. In order to convert an input to a memory element, it is indeed sufficient to multiply it by a memory value containing 1. The latter corresponds to a subtractive secret-sharing of 1, e.g. $y_1 = 1$ and $y_0 = 0$ and a subtractive secret-sharing of s, which was provided to the parties by the initial setup. Outputting the value of a memory wire M_z is even simpler, the parties just broadcast their share of z reducing it modulo n_{out}. By subtracting the two messages modulo n_{out}, the players can obtain the final result of the computation.

On the bound on the values of the wires. The correctness of the HSS scheme described above relies on the assumption that $|x \cdot y \cdot s| \ll t$ for every multiplication. If this condition is not satisfied, there is a non-negligible probability that the secret-sharing over $\mathbb{Z}_t$ obtained as result cannot be converted into an integer secret-sharing of the same value. Observe, anyway, that denoting by B the bound of the RMS circuit, $|x \cdot y \cdot s| \leq B \cdot 2^{\mathsf{len}_{\mathsf{sk}}}$, so, in order to circumvent the problem, we can choose the parameters of the NIDLS framework so that $B \cdot 2^{\mathsf{len}_{\mathsf{sk}}} \cdot 2^{\lambda} < t$.

Theorem 1. *If the DDH assumption and the small exponent assumption hold in the NIDLS framework and* F *is a secure PRF outputting values in* $\mathbb{Z}_t$*, the construction in Fig. 2 is a correct and secure HSS scheme for RMS circuits with bound* $B < t/2^{\mathsf{len}_{\mathsf{sk}}+\lambda}$*. The ring where the computation takes place is* $R = \mathbb{Z}_{n_{\mathsf{out}}}$*. Assuming* $n_{\mathsf{out}} < B$*, the input space is* $\mathcal{I} = R$*.*

We prove Theorem 1 in the full version of the paper [ADOS22, Section 5.2].

5.3 Implementing the Setup Using One Round.

The HSS scheme described in Fig. 2 relies on a setup producing a NIDLS ElGamal public key and a subtractive secret-sharing over the integers of the private counterpart. One of the main goals of this work is to improve upon the results of [OSY21] by removing the need for trusted dealers. In this section, we therefore explain how the parties can setup the HSS material in one round. The protocol, which is formally described in Fig. 3, relies on a CRS providing the parties with the parameters of the NIDLS framework and a PRF key k. When the framework is instantiated over class groups, the generation of the CRS does not need any trusted dealer. Indeed, the parties just need to produce public, random coins and input them into the algorithm producing the CRS. In the random oracle model, this procedure can be performed non-interactively. In [OSY21], the HSS scheme was based on Paillier. Since the associated group is described by an RSA modulo N where $\varphi(N)$ needs to remain secret, designing an efficient setup for the HSS scheme without relying on trusted dealers is a challenging task in that case.

Our setup protocol is very simple. Each party just generates a NIDLS ElGamal key pair, publishing the public counterpart. The parties then output the quotient between the two public keys and their respective secret key.

Function $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$

1. Compute $\big(\mathsf{par}, g, \rho, \ell_{\mathsf{sk}}, \mathsf{pk}, k, (s_0, s_1)\big) \xleftarrow{R} \mathsf{Setup}(\mathbb{1}^\lambda)$
2. Output $(\mathsf{par}, g, \rho, \ell_{\mathsf{sk}}, \mathsf{pk}, s_i, k)$ to every party P_i.

Protocol $\Pi_{\mathsf{HSS\text{-}Setup}}$

CRS:

1. Let $\mathsf{par} := (G, F, H, f, t, \ell, \mathsf{aux}) \xleftarrow{R} \mathsf{Gen}(\mathbb{1}^\lambda)$ and ℓ_{sk} be the parameter for the small-exponent assumption.
2. $(g, \rho) \xleftarrow{R} \mathcal{D}(\mathbb{1}^\lambda, \mathsf{par})$
3. $k \xleftarrow{R} \{0,1\}^\lambda$
4. Output $(\mathsf{par}, \ell_{\mathsf{sk}}, g, \rho, k)$

PROCEDURE:

1. Every party P_i samples $s_i \xleftarrow{R} [\ell_{\mathsf{sk}}]$
2. Every party P_i sends $\mathsf{pk}_i \leftarrow g^{s_i}$ to P_{1-i}
3. Every party P_i outputs $\mathsf{pk} \leftarrow \mathsf{pk}_1/\mathsf{pk}_0$, s_i and k.

Fig. 3. The HSS setup functionality and a one-round protocol implementing it.

Theorem 2. *The protocol $\Pi_{\mathsf{HSS\text{-}Setup}}$ implements the functionality $\mathcal{F}_{\mathsf{HSS\text{-}Setup}}$ against a semi-honest adversary with perfect security.*

Proof. Suppose that P_i is corrupted. The simulator receives $(\mathsf{par}, g, \rho, \ell_{\mathsf{sk}}, \mathsf{pk}, s_i, k)$ from the functionality. It can then simulate the CRS by providing the adversary with $(\mathsf{par}, \ell_{\mathsf{sk}}, g, \rho, k)$. The view of P_i is perfectly simulated by sending s_i and $\mathsf{pk} \cdot g^{s_i}$ if $i = 0$ or g^{s_i}/pk if $i = 1$. Observe that the output of P_{1-i} is consistent with the elements sent to the adversary. $\square$

6 Public-Key PCFs and One-Round VOLE Protocol without Trusted Setup

In [OSY21], the authors designed a one-round VOLE protocol based on the Paillier cryptosystem and the NIDLS problem on the underlying group. A VOLE protocol involves two parties, the input of the first one is a element in a ring R, the input of the second party is a R-vector $\boldsymbol{a}$. The output of the protocol consists of an additive secret-sharing of the product $x \cdot \boldsymbol{a}$.

We now present a version of such protocol in the NIDLS framework (see Fig. 5). By generalising the techniques to a more abstract setting, we are able to

Function $\mathcal{F}_{\mathsf{VOLE}}$

INITIALISATION: The functionality waits for a value $t \in \mathbb{N}$ from the adversary.
EVALUATION: On input $x \in \mathbb{Z}_t$ from P_0 and $\boldsymbol{a} \in \mathbb{Z}_t^m$ from P_1, the functionality sends m to the adversary.

- If both parties are honest, $\mathcal{F}_{\mathsf{VOLE}}$ samples $\boldsymbol{y}_0 \xleftarrow{R} \mathbb{Z}_t^m$ and sets $\boldsymbol{y}_1 \leftarrow \boldsymbol{a} \cdot x - \boldsymbol{y}_0$. Then, it outputs $\boldsymbol{y}_i$ to P_i for every $i \in \{0, 1\}$.
- If P_i is corrupt, $\mathcal{F}_{\mathsf{VOLE}}$ waits for $\boldsymbol{y}_i \in \mathbb{Z}_t^m$ from the adversary and sets $\boldsymbol{y}_{1-i} \leftarrow \boldsymbol{a} \cdot x - \boldsymbol{y}_i$. Then, it outputs $\boldsymbol{y}_{1-i}$ to P_{1-i}.

Function $\mathcal{F}_{\mathsf{NIKE}}$

If both parties are honest, sample $k \xleftarrow{R} \{0, 1\}^\lambda$ and output it to all the parties.
If one party is corrupted, wait for $k \in \{0, 1\}^\lambda$ from the adversary and output it to the other party.

Fig. 4. The NIKE and vector-OLE functionalities

leverage the properties of the various instantiations. In the case of class groups, that allows us to not rely on any trusted setup. In order to achieve this goal, we had to slightly modify the CRS used by the protocol. In [OSY21], the latter consisted of a pair of group elements (g, C) where $C = g^r$ for some unknown r. In order to avoid trusted setups, we now need to provide the parties with the randomness used for the generation of the CRS. Unfortunately, in class groups, such randomness would leak the value of r to the adversary, compromising security. In order to circumvent the problem, in this work, g and C are sampled independently using $\mathcal{D}(\mathbb{1}^\lambda)$, so with high probability $C \notin \langle g \rangle$. We prove security by relying on the DXDH assumption.

The construction makes use of a non-interactive key exchange functionality $\mathcal{F}_{\mathsf{NIKE}}$ (see Fig. 4). The latter provides the parties with a random PRF key $k \in \{0, 1\}^\lambda$. When one of the parties is corrupt, the functionality lets the adversary choose k, forwarding it to the honest party. It is possible to implement $\mathcal{F}_{\mathsf{NIKE}}$ in one round using NIKE constructions such as Diffie-Hellman.

Correctness. To understand why the protocol works, observe that

$$D^{r_1^i} \cdot E^{a_i} = g^{r_0 \cdot r_1^i} \cdot f^{x \cdot a_i} \cdot C^{r_0 \cdot a_i} = f^{x \cdot a_i} \cdot A_i^{r_0}.$$

In other words, for every index i, the elements $D^{r_1^i} \cdot E^{a_i}$ and $A_i^{-r_0}$ are a multiplicative secret-sharing of $f^{x \cdot a_i}$. Using DDLog for every $i \in [m]$, the parties are therefore able to obtain an additive secret-sharing of $x \cdot \boldsymbol{a}$ without any additional interaction.

Protocol $\mathit{\Pi}_{\mathsf{VOLE}}$

Inputs: The first party P_0 has input $x \in \mathbb{Z}_t$. The other party P_1 has input $\boldsymbol{a} \in \mathbb{Z}_t^m$ for some $m \in \mathbb{N}$.
Setup $\mathsf{Setup}(1\!\!1^\lambda)$:

1. $\mathsf{par} := (G, F, H, f, t, \ell, \mathsf{aux}) \overset{R}{\leftarrow} \mathsf{Gen}(1\!\!1^\lambda)$
2. $(g, \rho_0) \overset{R}{\leftarrow} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
3. $(C, \rho_1) \overset{R}{\leftarrow} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
4. If $g = C$, go to step 3.
5. Output $(\mathsf{par}, g, \rho_0, C, \rho_1)$

Procedure:

1. The parties call $\mathcal{F}_{\mathsf{NIKE}}$ to obtain a key $k \in \{0,1\}^\lambda$.
2. $\forall i \in [m]:\quad P_1$ sends $A_i \leftarrow g^{r_1^i} \cdot C^{a_i}$ where $r_1^i \overset{R}{\leftarrow} [\ell]$.
3. P_0 sends $(D, E) \leftarrow (g^{r_0}, f^x \cdot C^{r_0})$ where $r_0 \overset{R}{\leftarrow} [\ell]$.
4. P_1 outputs $\boldsymbol{y}_1$ where $\boldsymbol{y}_1[i] \leftarrow \mathsf{DDLog}_{\mathsf{par}}(D^{r_1^i} \cdot E^{a_i}) + \mathsf{F}_k(i)$ for every $i \in [m]$.
5. P_0 outputs $\boldsymbol{y}_0$ where $\boldsymbol{y}_0[i] \leftarrow \mathsf{DDLog}_{\mathsf{par}}(A_i^{r_0}) - \mathsf{F}_k(i)$

Fig. 5. A one-round VOLE protocol based on the NIDLS framework.

Security. At first glance, it might seem that the security of the protocol follows from the fact that the A_i's are Pedersen commitments with respect to (g, C). However, note that the element C is *not guaranteed to be* in the cyclic group generated by g. As a consequence, A_i does not hide the input a_i with information-theoretic security and we need instead to rely on a computational assumption. The same happens also in step 3, where C plays the role of the public key in an NIDLS ElGamal encryption. However, again, C is not guaranteed to belong to $\langle g \rangle$. Therefore, we need to argue for security in a different way. To solve both issues, we use the DXDH assumption (see Definition 5).

Observe that under the DXDH assumption, g^r looks like C^s even when the randomness used for the generation of the CRS is known. As a consequence, no adversary can distinguish $A_i = g^{r_1^i} \cdot C^{a_i}$ from $C^s \cdot C^{a_i}$. The latter contains no information about a_i. The privacy of x is instead preserved as $(D, E) = (g^{r_0}, f^x \cdot C^{r_0})$ is indistinguishable, under our assumption, from $(C^s, f^x \cdot C^r)$. Since the distribution $\mathcal{D}$ outputs an element C such that $f \in \langle C \rangle$, the pair $(C^s, f^x \cdot C^r)$ hides all the information about x. The proof of the next theorem is omitted, as it easily follows from the arguments sketched above.

Theorem 3. *If the DXDH assumption holds and* F *is a secure PRF outputting pseudorandom elements in* $\mathbb{Z}_t$*, the protocol* $\mathit{\Pi}_{\mathsf{VOLE}}$ *UC implements the functionality* $\mathcal{F}_{\mathsf{VOLE}}$ *against a semi-honest adversary in the* $\mathcal{F}_{\mathsf{NIKE}}$*-hybrid model.*

6.1 Public-Key PCFs without Trusted Setup.

In [OSY21], Orlandi *et al.* present PCFs for vector-OLE and OT based on Paillier and the Goldwasser-Micali cryptosystem respectively (see the full version of the paper [ADOS22, Fig. 8-9]). The interesting property of both constructions is that, thanks to the one-round VOLE protocol of [OSY21], the PCF keys can be set up using only one round of interaction and low-communication in the output size. For this reason, the authors introduced the notion of public-key PCF to refer to them.

On the downside, as we mentioned in the previous subsection, the one-round VOLE protocol of [OSY21] needs a trusted setup. The issue is immediately inherited by the public-key PCFs. Now, by plugging our new VOLE protocol, we obtain public-key PCFs with no need for trusted setups. We describe the resulting protocols in the full version of the paper [ADOS22, Section 6.1].

On the need for the hardness of factoring. The security of both our public-key PCFs still relies on the hardness of factoring. This requirement is inherited from the original PCFs of [OSY21]. At first, it may seem possible to generalise the two constructions to the NIDLS framework, potentially obtaining public-key PCFs based on class groups only. Unfortunately, this turns out to be false.

Indeed, in the public-key PCFs for VOLE and OT, we need to non-interactively sample random ciphertexts without leaking any information about the plaintext to P_1. For Paillier, this is not a problem as any element in $\mathbb{Z}_{N^2}^\times$ is a valid encryption. For Goldwasser-Micali instead, it is sufficient to sample a random element in $\mathbb{Z}_N$ with Jacobi symbol 1. Now, if we try to move the constructions to class groups, we need to use the ElGamal cryptosystem. By modifying the PCF keys and using techniques as in the HSS scheme (see Section 5), it is still possible for the parties to non-interactively obtain an additive secret-sharing of $a \cdot x$ given the encryption of a random a. The issue is that the only known way to sample such encryption is to directly encrypt a (not every pair of elements in the class group is an ElGamal ciphertext). That would leak the value of a to P_1.

7 Actively Secure Public-Key PCFs

In addition to requiring a trusted setup, the public key PCFs in [OSY21] achieve security in the semi-honest setting only. In this section, we explain how to upgrade the constructions described in Section 6 to active security, while preserving, at the same time, their round-complexity properties, namely that the parties need to speak only once. When the NIDLS framework is instantiated over class groups, the constructions do not need any trusted setup.

The particular interaction pattern limits the techniques we can rely on. For instance, we cannot perform checks that verify the correctness of the outputs, as that would require an additional round of interaction after the outputs are derived. For this reason, we develop NIZKs for our framework which might be of independent interest. We start presenting some building blocks (commitments in Section 7.1 and ZK proofs in Section 7.2). Then, we describe an actively secure public-key PCF for vector-OLE (Section 7.3). In the full version of the paper [ADOS22, Section 7.4], we also present an active public-key PCF for OT.

7.1 An Integer Commitment Scheme in the NIDLS Framework

Our NIZKs follow a commit-and-prove approach. Notice that in order to achieve active security, party P_0 has to prove that its input to the one-round vector-OLE protocol is the private key associated with the RSA modulo N. For this reason, we need to prove particular number-theoretic relations for which commitment schemes based on modular rings such as $\mathbb{Z}_t$ are not really suited.

Recall that, in the NIDLS framework, determining the order of the group from its parameters is assumed to be hard. This property crucially allows us to design integer commitment schemes. This fact was already noticed for class groups by Couteau *et al.* [CKLR21]. In this work, we adopt a generalisation of their construction to the NIDLS framework (see Fig. 6), basing however its security on the DXDH assumption. As we discuss at the end of this section, despite the claims in [CKLR21], their construction is not compatible with trustless setup.

```
Integer commitment scheme

Crs:
  1. $\mathsf{par}' := (G, F, H, f, t, \ell, \mathsf{aux}) \overset{R}{\leftarrow} \mathsf{Gen}(1\!\!1^\lambda)$
  2. $(g, \rho_0) \overset{R}{\leftarrow} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
  3. $(C, \rho_1) \overset{R}{\leftarrow} \mathcal{D}(1\!\!1^\lambda, \mathsf{par})$
  4. If $C = g$, go to step 3.
  5. Output $\mathsf{par} := (\mathsf{par}', g, \rho_0, C, \rho_1)$

Commitment: $\mathsf{Commit}(\mathsf{par}, x)$
  1. $r \overset{R}{\leftarrow} [\ell]$
  2. Output the commitment $X \leftarrow C^x \cdot g^r$ and the opening information $r$.

Verification: $\mathsf{Verify}(\mathsf{par}, X, x, r)$
  1. If $X = C^x \cdot g^r$ output 1 otherwise output 0.
```

Fig. 6. Integer commitment scheme in the NIDLS framework.

Theorem 4. *If the DXDH assumption and the weak hidden order assumption hold, the construction in Fig. 6 is a hiding and binding integer commitment scheme. Moreover, the scheme is linearly homomorphic.*

Proof. It is straightforward to see that the construction is correct and linearly homomorphic.

Binding. The proof for binding is made interesting by the fact that C is not (necessarily) an element in the group generated by g. Suppose that we have an adversary that breaks binding e.g., after being provided with the parameters, the adversary returns (x, r) and (y, s) with $x \neq y$ such that $C^x g^r = C^y g^s$, and therefore $C^{x-y} = g^{s-r}$. Let $\alpha := x - y \neq 0$ and $\beta = s - r$. Since the order of the group is unknown we cannot invert these elements. Instead, we resort to the DXDH assumption, which implies the following claim:

Claim. Assume there exists an adversary $\mathcal{A}$ that, on input (g, C), returns (α, β) with $C^\alpha = g^\beta$ and $\alpha \neq 0$. Then, with overwhelming probability over $u, v \overset{R}{\leftarrow} [\ell]$, it holds that:

$$(g^u)^\alpha = (g^v)^\beta \tag{1}$$

Proof (of claim). The reduction is given a DXDH tuple (g, C, g^u, T) where T is either C^u or g^v for random $u, v \overset{R}{\leftarrow} [\ell]$, and feeds (g, C) to $\mathcal{A}$. Now the reduction concludes that $T = C^u$ when

$$(T)^\alpha = (g^v)^\beta \tag{2}$$

or $T = g^v$ otherwise. Note that if $T = C^u$ then Equation 2 is trivially true. Thus if $(g^u)^\alpha \neq (g^v)^\beta$ the reduction correctly distinguished between DXDH tuple and non-DXDH tuples. □

We now go back to the proof of the binding property and argue that under the weak hidden order assumption, no adversary can output α, β such that Equation 1 holds for random (u, v). We first rewrite $(g^u)^\alpha = (g^v)^\beta$ as $u \cdot \alpha \equiv v \cdot \beta \bmod \mathsf{ord}(g)$. We argue the following:

Claim. Let α, β be such that

$$u \cdot \alpha \equiv v \cdot \beta \bmod \mathsf{ord}(g)$$

with overwhelming probability for uniform $u, v \overset{R}{\leftarrow} [\ell]$. Then $\mathsf{ord}(g) | \alpha$.

Proof (of claim). For the sake of contradiction assume that this is not the case, e.g., $\mathsf{ord}(g) \nmid \alpha$. Then there is a non-negligible probability that $u \cdot \alpha \not\equiv v \cdot \beta \bmod \mathsf{ord}(g)$. Indeed, let p be a prime that divides $\mathsf{ord}(g)$ but not α, it must hold that $u \equiv v \cdot \beta \cdot \alpha^{-1} \bmod p$. This happens with probability $1/p < 1/2$, so it must be that α is a multiple of $\mathsf{ord}(g)$. □

We have reached a contradiction. Indeed, under the weak hidden order assumption, no adversary can output α such that $g^\alpha = 1$.

Hiding. We show that no adversary can distinguish a commitment to x_0 from a commitment to x_1. Indeed by the DXDH assumption, (g, C, g^r, C^r) with r uniform in $[\ell]$, is indistinguishable from (g, C, C^s, C^r) with s again uniform in $[\ell]$. Thus $C^{x_b} \cdot g^r$ is indistinguishable from $C^{x_b} \cdot C^s$. From the way ℓ is chosen, C^s is statistically close to the uniform distribution over $\langle C \rangle$. So, as commitments to both x_0, x_1 are indistinguishable from random elements in $\langle C \rangle$, no adversary can distinguish between a commitment to x_0 and a commitment to x_1. $\square$

In [CKLR21], the authors proved the security of this commitment scheme in the class group setting by relying on the *subgroup indistinguishability* assumption. The latter states that no PPT adversary can distinguish between a pair of random elements (g, C) both sampled according to $\mathcal{D}$ and a pair (g, g^s) where s is uniform over $[\ell]$. Despite what the authors claim, this assumption is not sufficient to prove security when we do not rely on a trusted dealer for the generation of the CRS. Indeed, in order to remove trusted setups, we need to provide the parties with the random coins used for the generation of the CRS. That prevents us from substituting C with g^s in the security proofs. The reason is that the distribution $\mathcal{D}$ is, surprisingly, not invertible over class groups. Specifically, given $C \in \mathsf{Supp}(\mathcal{D})$, it is hard to find a bit string r such that $\mathcal{D}(1\!\!1^\lambda; r) = C$.

7.2 Zero-Knowledge Proofs in the NIDLS Framework

We describe how to build useful ZK-proofs in the NIDLS framework such as: range proofs, multiplication proofs, proofs of knowledge of openings and proofs of commitment to the plaintext. In particular, we build sigma protocols that use the NIDLS framework in a black-box way, independently of its instantiation. Thus, our proofs do not need to express operations in the NIDLS framework as circuits. Since these tools are all based on fairly standard techniques, we will only give a brief overview and direct the reader to the full version of the paper [ADOS22, Appendix A] for more details.

Proof of knowledge of openings. Π_{com} allows to convince a verifier holding a commitment X that the prover knows integers x and r such that $X = C^x \cdot g^r$.

Compared to standard Σ-protocols for proving knowledge of a (Pedersen) commitment in a prime order group, we need two major changes: First, all the computation between scalars is done over the integers (since the order of the group is unknown) and therefore, the random strings chosen in the first round must be larger than an upper bound on the witness (x, r). Second, we can only use binary challenges: this is due to the fact that, again, the order of the group is unknown and therefore, we cannot invert the challenge when extracting the witness in the special soundness property. Thus, we need to repeat the proofs λ times. Note that for most of our instantiations there usually are ways around this issue, mostly relying on instantiation-dependent assumptions (such as the strong root problem and the low order assumption for class groups). However, those do not carry over to our general framework.

Multiplication proofs. Π_{mult} allows to convince a verifier with commitments X, Y and Z, that the prover knows $x, y, z \in \mathbb{Z}$ and $r_1, r_2, r_3 \in \mathbb{Z}$ such that $X = C^x \cdot g^{r_1}$, $Y = C^y \cdot g^{r_2}$, $Z = C^z \cdot g^{r_3}$ and $z = x \cdot y$. We construct Π_{mult} by adapting the protocol of [DF02] to our framework, similarly to what we did for Π_{com}.

Range proofs. Π_{range} allows to convince a verifier holding a commitment X and a bound $B \in \mathbb{N}$ that the prover knows $x, r \in \mathbb{Z}$ such that $x \in [0, B]$ and $X = C^x \cdot g^r$. Our protocol is based on a technique by Groth [Gro05], who observed that

$$x \in [0, B] \quad \Longleftrightarrow \quad \exists x_1, x_2, x_3 \in \mathbb{Z} \quad \text{s.t. } 1 + 4x \cdot (B - x) = x_1^2 + x_2^2 + x_3^2.$$

The protocol can be therefore constructed exploiting multiplication proofs just introduced and the linearity of the commitment.

We remark that in [CKLR21], the authors designed a range proof for our commitment scheme in the class group setting. Their solution never relies on binary challenges, so its efficiency is better by a factor of λ. However, their construction is only proven secure when the CRS is generated by a trusted dealer. This is due to the issue described at the end of Section 7.1.

Proof of commitment to the plaintext. Π_{plain} can be used to convince a verifier holding group elements D, E, X that the prover knows $x, r, s \in \mathbb{Z}$ such that $X = C^x \cdot g^s$, $D = g^r$ and $E = f^x \cdot C^r$. The protocol uses standard techniques adapted to our framework as sketched for Π_{com}.

7.3 Actively Secure Public-Key PCF for Vector-OLE

In the semi-honest public-key PCF for vector-OLE (Fig. 5 and [ADOS22, Fig. 8]), the only message sent by party P_0 consists of an RSA modulo N and a pair of groups elements D, E where $D = g^{r_0}$ and $E = f^d \cdot C^{r_0}$. Here, the exponent d represents the Paillier private key associated with the RSA modulo N, whereas g and C are groups elements described in the CRS. We recall that d is the only element in $[0, N \cdot \varphi(N) - 1]$ satisfying $d \equiv 0 \bmod \varphi(N)$ and $d \equiv 1 \bmod N$.

The only message sent by party P_1 is instead $A := C^x \cdot g^{r_1}$. In order for the construction to be correct, the value of x needs to be smaller than $2^\lambda \cdot 2^{\mathsf{len}_N}$.

An active adversary can always deviate from the protocol and send malformed material. For this reason, it is fundamental that our NIZKs prove the well-formedness of the messages of the parties. In the case of P_1, the task is rather simple. Using the Fiat-Shamir heuristic, we can indeed convert Π_{range} into the NIZK we are looking for. Proving the well-formedness of P_0's message is however more challenging.

Proving the well-formedness of P_0's message. As usual we first design a public coin honest-verifier zero-knowledge proof and then convert it into a NIZK by applying the Fiat-Shamir heuristic. Our protocol makes use of a public-coin HVZK $\Pi_{\mathsf{semiprime}}$ for proving that the RSA modulo N is the product of two distinct primes p and q. Moreover, $\Pi_{\mathsf{semiprime}}$ proves that $\gcd\big(N, \varphi(N)\big) = 1$. Such protocol can be found e.g., in [GRSB19].

The main idea of our protocol is as follows: the prover commits to d, the primes p and q and integers k_1 and k_2 satisfying $d = k_1 \cdot \varphi(N)$ and $d = k_2 \cdot N + 1$. We denote the five commitments by Z, X_1, X_2, Y_1 and Y_2 respectively. The parties run $\Pi_{\mathsf{semiprime}}$ to verify that N is semiprime. By relying on Π_{mult}, the prover also shows that X_1 and X_2 are commitments to a factorisation of N. Furthermore, using Π_{range}, the verifier checks that the value committed in X_1 belongs to $[2, N-1]$ (this is done by showing that $C^{-2} \cdot X_1$ is a commitment to a value in $[0, N-3]$). In this way, it is sure that the prover committed to a proper factorisation and not just $N \cdot 1$. Now, the verifier is also certain that $W := C^N \cdot X_1^{-1} \cdot X_2^{-1} \cdot C$ is a commitment to $N - p - q + 1 = \varphi(N)$. Next, using Π_{range}, the prover shows that the value committed in Y_1 belongs to $[0, N-1]$. Using Π_{com}, it also proves the knowledge of opening for Y_2. The verifier also checks that Y_1 is a commitment to $d/\varphi(N)$. This is done by running Π_{mult} on Y_1, W and Z. If the check passes, the verifier is also sure that the value committed in Z belongs to $[0, N \cdot \varphi(N) - 1]$. In the end, the prover shows that Y_2 is a commitment to $(d-1)/N$ by proving that $Z \cdot C^{-1} \cdot Y_2^{-N}$ opens to 0. Finally, the prover uses Π_{plain} to convince the verifier that the values hidden in Z and in (D, E) coincide. The formal description of the protocol, which we call Π_{Paillier}, and the proof of the following theorem are in the full version of the paper [ADOS22, Section 7.3].

Theorem 5. *Let $\Pi_{\mathsf{semiprime}}$ be a honest-verifier zero-knowledge public-coin proof proving that N is the product of two distinct primes and $\gcd\big(N, \varphi(N)\big) = 1$. If the commitment scheme in Fig. 6 is hiding and binding, the construction Π_{Paillier} (see [ADOS22, Fig. 11]) is a complete, special-sound public-coin proof for the relation*

$$\mathcal{R}_{\mathsf{Paillier}} := \left\{ (D, E, N), (d, p, q, r) \left| \begin{array}{l} N = p \cdot q, \ \text{where } p, q \text{ are positive primes} \\ \gcd\big(N, \varphi(N)\big) = 1 \\ D = g^r, E = f^d \cdot C^r \\ d \equiv 0 \bmod \varphi(N) \\ d \equiv 1 \bmod N \\ 0 \le d < N \cdot \varphi(N) \end{array} \right. \right\}$$

Moreover, when $r \in [\ell]$, the proof is honest-verifier zero-knowledge.

Deploying the NIZKs to obtain active security. We can finally present our active public key PCF for vector-OLE. The construction, called $\Pi^{\mathsf{Active}}_{\mathsf{VOLE}}$, is described in Fig. 7.

We prove that the pk-PCF protocol implements the random vector-OLE functionality $\mathcal{F}_{\mathsf{r\text{-}VOLE}}$ (see Fig. 8) in the UC model. $\mathcal{F}_{\mathsf{r\text{-}VOLE}}$ is a functionality that, during the initialisation, samples a random RSA modulo N and a value $x \in \mathbb{Z}_N$, which outputs to P_1. Upon any request for a vector-OLE tuple, the

Active PK-PCF for VOLE $\Pi^{\mathsf{Active}}_{\mathsf{VOLE}}$

Let F be a PRF. Let len_N denote the length of the Paillier modulo and let t, the order of the NIDLS group, be greater than $2^\lambda \cdot 2^{2\mathsf{len}_N} \cdot 2^{\lambda+\mathsf{len}_N}$.

Initialisation:

1. The parties initialise $\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}$ obtaining $\mathsf{par} := (\mathsf{par}', g, \rho_0, C, \rho_1)$.
2. The parties call $\mathcal{F}_{\mathsf{NIKE}}$ to obtain a PRF key k.
3. P_0 computes $(N, d) \overset{R}{\leftarrow} \mathsf{Paillier.Gen}(1\!\!1^\lambda)$ where $N = p \cdot q$.
4. P_1 samples $x \overset{R}{\leftarrow} [B]$ where $B := 2^{\lambda+\mathsf{len}_N}$.
5. P_0 samples $r_0 \overset{R}{\leftarrow} [\ell]$ and sets $D \leftarrow g^{r_0}$, $E \leftarrow f^d \cdot C^{r_0}$.
6. P_0 sends N, D, E.
7. P_1 samples $r_1 \overset{R}{\leftarrow} [\ell]$ and computes $A \leftarrow C^x \cdot g^{r_1}$
8. P_1 sends A.
9. The parties call $\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}$ with input $(\mathsf{Paillier}, D, E, N)$. P_0 inputs also (p, q, r_0). The parties abort if the functionality outputs 0 or if $N > 2^{\mathsf{len}_N}$.
10. The parties call $\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}$ with input (range, A, B). P_1 inputs also (x, r_1). The parties abort if the functionality outputs 0.
11. The parties query (A, D, E, N) to the random oracle and obtain a random $u \in \mathbb{Z}_t$ as a reply.
12. P_0 computes $v_0 \leftarrow \mathsf{DDLog}_{\mathsf{par}}(A^{r_0}) - u \bmod t$
13. P_1 computes $v_1 \leftarrow \mathsf{DDLog}_{\mathsf{par}}(D^{r_1} \cdot E^x) + u \bmod t$
14. P_0 stores $\mathsf{k}_0 \leftarrow (N, k, y_0 := -v_0, d)$.
15. P_1 stores $\mathsf{k}_1 \leftarrow (N, k, y_1 := v_1, x \bmod N)$.

Evaluation: Query the label id to the oracle. Let $\mathsf{ct} \in \mathbb{Z}^\times_{N^2}$ be the response:

1. P_0 computes $a \leftarrow \mathsf{Paillier.Dec}(d, \mathsf{ct}) \bmod N$.
2. Each P_i computes $z_i \leftarrow (-1)^{1-i} \cdot \mathsf{DDLog}_{\mathsf{Paillier}}(\mathsf{ct}^{y_i}) + \mathsf{F}_k(\mathsf{ct}) \bmod N$.
3. P_0 outputs (a, z_0), P_1 outputs $(x \bmod N, z_1)$.

Fig. 7. Active public-key PCF for vector-OLE

functionality samples a random $a \in \mathbb{Z}_N$ and computes a subtractive secret-sharing of $z_1 - z_0 = a \cdot x$ over $\mathbb{Z}_N$. Then, $\mathcal{F}_{\mathsf{r\text{-}VOLE}}$ outputs (a, z_0) to P_0 and z_1 to P_1. If one of the parties is corrupted, the functionality let the adversary choose the outputs of the corrupt player, then it samples the outputs of the honest party at random conditioned on $z_1 = z_0 + a \cdot x$. Moreover, if P_0 is corrupted, the functionality lets the adversary select the RSA modulo N. When P_1 is corrupt, instead, $\mathcal{F}_{\mathsf{r\text{-}VOLE}}$ lets the adversary choose x after providing it with N.

The resources. The protocol $\Pi^{\mathsf{Active}}_{\mathsf{VOLE}}$ relies on the non-interactive key-exchange functionality $\mathcal{F}_{\mathsf{NIKE}}$ (see Fig. 4) and a ZK functionality $\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}$ (see Fig. 9). The former provides the parties with a random PRF key $k \in \{0,1\}^\lambda$. When one of the parties is corrupt, the functionality lets the adversary choose k, forwarding

Function $\mathcal{F}_{\textsf{r-VOLE}}$

Initialisation:

- If both parties are honest, generate $(N, p, q) \xleftarrow{R} \textsf{Paillier.Gen}(\mathbb{1}^\lambda)$ and sample $x \xleftarrow{R} \mathbb{Z}_N$.
- If P_0 is corrupt, wait for N from the adversary and sample $x \xleftarrow{R} \mathbb{Z}_N$. If the adversary sends $\perp$, abort.
- If P_1 is corrupt, generate $(N, p, q) \xleftarrow{R} \textsf{Paillier.Gen}(\mathbb{1}^\lambda)$, send N to the adversary to the adversary and wait for $x \in \mathbb{Z}_N$ as a reply. If the adversary sends $\perp$, abort.

Evaluation: On input a fresh label $\textsf{id}$ from an honest party P_i.

- If both parties are honest, the functionality samples $a, z_0 \xleftarrow{R} \mathbb{Z}_N$ and sets $z_1 \leftarrow a \cdot x - z_0$. Then, it sets $R_0 \leftarrow (a, z_0)$ and $R_1 \leftarrow (x, z_1)$. $\mathcal{F}_{\textsf{r-VOLE}}$ outputs R_i to P_i and stores $(\textsf{id}, 1-i, R_{1-i})$.
- If $i = 1$ and P_0 is corrupted, the functionality waits for $a, z_0 \in \mathbb{Z}_N$ from the adversary and sets $z_1 \leftarrow a \cdot x - z_0$. Then, it outputs (x, z_1) to P_i.
- If $i = 0$ and P_1 is corrupted, the functionality waits for $z_1 \in \mathbb{Z}_N$ from the adversary, samples $a \xleftarrow{R} \mathbb{Z}_N$ and computes $z_0 \leftarrow a \cdot x - z_1$. Then, it outputs (a, z_0) to P_i.

If $\textsf{id}$ is not fresh, retrieve the triple $(\textsf{id}, i, R_i)$ and output R_i to P_i.

Fig. 8. The random vector-OLE functionality

it to the honest party. It is possible to implement $\mathcal{F}_{\textsf{NIKE}}$ in one round using NIKE constructions such as Diffie-Hellman, augmenting them with NIZKs to achieve security against an active adversary.

The functionality $\mathcal{F}_{\textsf{NIDLS-ZK}}$ is instead used to prove statements for a fixed set of NP relations. We can assume that this set includes range proofs and $\mathcal{R}_{\textsf{Paillier}}$. Upon initialisation, $\mathcal{F}_{\textsf{NIDLS-ZK}}$ outputs the parameters of the NIDLS framework, including the random coins used for their generation. When $\mathcal{F}_{\textsf{NIDLS-ZK}}$ is provided with a statement x for one of the supported NP relations, the functionality waits for the prover to provide the corresponding witness w. If the verification fails, $\mathcal{F}_{\textsf{NIDLS-ZK}}$ outputs 0 to both parties, otherwise, it outputs 1. The functionality $\mathcal{F}_{\textsf{NIDLS-ZK}}$ is also equipped with a different predicate for each supported NP-relation. Such predicate makes sure that the witness satisfies the properties for zero-knowledge. If that is not the case, the w is leaked to the adversary.

Note that Fiat-Shamir NIZKs, including the ones we designed, do not implement the functionality $\mathcal{F}_{\textsf{NIDLS-ZK}}$ in the UC model. Indeed, in order to extract the witness w, we need to rewind the adversary and this operation is incompatible with UC. Using Fiat-Shamir NIZKs to implement $\mathcal{F}_{\textsf{NIDLS-ZK}}$ is, however, a common practice, which is considered secure. Moreover, the resulting protocols

Function $\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}$

Let $\mathcal{U}$ be a finite set of NP relations. Let P_{L} be a predicate corresponding to the relation $\mathcal{R}_{\mathsf{L}} \in \mathcal{U}$.

Initialisation:

1. $\mathsf{par}' := (G, F, H, f, t, \ell, \mathsf{aux}) \overset{R}{\leftarrow} \mathsf{Gen}(1\!\!1^{\lambda})$
2. $(g, \rho_0) \overset{R}{\leftarrow} \mathcal{D}(1\!\!1^{\lambda}, \mathsf{par})$
3. $(C, \rho_1) \overset{R}{\leftarrow} \mathcal{D}(1\!\!1^{\lambda}, \mathsf{par})$
4. If $C = g$, go to step 3.
5. Output $\mathsf{par} := (\mathsf{par}', g, \rho_0, C, \rho_1)$ to all the parties.

Verify:

On input an NP relation $\mathcal{R}_{\mathsf{L}} \in \mathcal{U}$ and a statement st from both parties and a witness w from only one of the parties, $\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}$ checks whether $(\mathsf{st}, w) \in \mathcal{R}_{\mathsf{L}}$. If that is the case, $\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}$ outputs 1 to all the parties, otherwise it outputs 0. If $P_{\mathsf{L}}(\mathsf{par}, w) = 0$, the functionality leaks w to the adversary.

Fig. 9. The NIDLS ZK functionality

can be proven secure in weaker models that allow sequential composability only. Finally, using standard techniques [DP92], it is still possible to adapt our NIZKs so that they implement $\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}$ in the UC model. The proof of the following theorem is in the full version of the paper [ADOS22, Section 7.3].

Theorem 6. *Let $\mathsf{len}_N(\lambda)$ be the length of the RSA modulo and assume that $t > 2^{2\lambda+3\mathsf{len}_N}$. Let F be a secure PRF outputting pseudorandom elements in $[2^{\lambda+\mathsf{len}_N}]$. If the DXDH assumption holds, the protocol $\Pi^{\mathsf{Active}}_{\mathsf{VOLE}}$ UC-implements the functionality $\mathcal{F}_{\mathsf{r\text{-}VOLE}}$ against an active adversary in the $(\mathcal{F}_{\mathsf{NIDLS\text{-}ZK}}, \mathcal{F}_{\mathsf{NIKE}})$-hybrid model with random oracle.*

Acknowledgments. The authors would like to thank Guilhem Castagnos and Fabien Laguillaumie for their crucial clarifications about the CL framework and class groups, Lasse Rønne Møller for important observations on the DXDH assumption and the anonymous reviewers for their feedback.

Research supported by: the Concordium Blockhain Research Center, Aarhus University, Denmark; the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC); the Independent Research Fund Denmark (DFF) under project number 0165-00107B (C3PO); the Aarhus University Research Foundation; and the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0085. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA). Distribution Statement “A” (Approved for Public Release, Distribution Unlimited).

References

ADOS22. Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An Algebraic Framework for Silent Preprocessing with Trustless Setup and Active Security. Cryptology ePrint Archive, Report 2022/363, 2022. https://eprint.iacr.org/2022/363.

BCG$^+$17. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS 2017*. ACM Press, October / November 2017.

BCG$^+$19. 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*, LNCS. Springer, Heidelberg, August 2019.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO 2016, Part I*, LNCS. Springer, Heidelberg, August 2016.

BHJL17. Fabrice Benhamouda, Javier Herranz, Marc Joye, and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. *Journal of Cryptology*, (2), April 2017.

BKS19. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In *EUROCRYPT 2019, Part II*, LNCS. Springer, Heidelberg, May 2019.

CJLN09. Guilhem Castagnos, Antoine Joux, Fabien Laguillaumie, and Phong Q. Nguyen. Factoring pq^2 with quadratic forms: Nice cryptanalyses. In *ASIACRYPT 2009*, LNCS. Springer, Heidelberg, December 2009.

CKLR21. Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. Efficient range proofs with transparent setup from bounded integer commitments. In *EUROCRYPT 2021, Part III*, LNCS. Springer, Heidelberg, October 2021.

CL15. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In *CT-RSA 2015*, LNCS. Springer, Heidelberg, April 2015.

DF02. Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT 2002*, LNCS. Springer, Heidelberg, December 2002.

DJ01. Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In *PKC 2001*, LNCS. Springer, Heidelberg, February 2001.

DJ03. Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP 03*, LNCS. Springer, Heidelberg, July 2003.

DP92. Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *33rd FOCS*. IEEE Computer Society Press, October 1992.

GMW86. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*. IEEE Computer Society Press, October 1986.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*. ACM Press, May 1987.

Gro05. Jens Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS 05*, LNCS. Springer, Heidelberg, June 2005.

GRSB19. Sharon Goldberg, Leonid Reyzin, Omar Sagga, and Foteini Baldimtsi. Efficient noninteractive certification of RSA moduli and beyond. In *ASIACRYPT 2019, Part III*, LNCS. Springer, Heidelberg, December 2019.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO 2008*, LNCS. Springer, Heidelberg, August 2008.

JL13. Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. In *EUROCRYPT 2013*, LNCS. Springer, Heidelberg, May 2013.

OSY21. Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021, Part I*, LNCS. Springer, Heidelberg, October 2021.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, LNCS. Springer, Heidelberg, May 1999.

PT00. Sachar Paulus and Tsuyoshi Takagi. A new public-key cryptosystem over a quadratic order with quadratic decryption time. *Journal of Cryptology*, (2), March 2000.

RS21. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In *CRYPTO 2021, Part III*, LNCS. Springer, Heidelberg, August 2021.

Tuc20. Ida Tucker. *Functional encryption and distributed signatures based on projective hash functions, the benefit of class groups. (Chiffrement fonctionnel et signatures distribuées fondés sur des fonctions de hachage à projection, l'apport des groupes de classe)*. PhD thesis, University of Lyon, France, 2020.