Malicious Homomorphic Secret Sharing with Applications to DV-NIZK and More

Pedro Capitão [*] Hila Dahari-Garbian [†] Lisa Kohl [‡] Zhe Li [§]

Abstract

Homomorphic Secret Sharing (CRYPTO 2016) allows a secret to be shared among two or more parties in such a way that the parties can locally evaluate a class of functions on their shares. Homomorphic secret sharing (HSS) schemes and their underlying techniques have facilitated a wide range of applications. To account for the fact that parties generating or evaluating the shares might act maliciously, variants of HSS schemes that allow detection of such malicious behavior have been introduced. However, all prior approaches of malicious HSS that capture the class of NC1 circuits either crucially rely on a random oracle or require an non-reusable setup.

In this work, we initiate the study of malicious public-key 2-party HSS in the standard model with reusable setup, where any malicious behavior during share generation and share evaluation can be detected. Towards constructing malicious HSS, we introduce the notion of homomorphic secret sharing with *robust linear reconstruction* (RLR-HSS) and show that this notion readily implies malicious HSS. We outline challenges in instantiating RLR-HSS due to the error present in all current HSS constructions not relying on SHE/FHE, and show how to overcome these using derandomization techniques by Dwork et al. (EUROCRYPT 2004). Finally, we show applications of malicious HSS to compact designated verifier non-interactive zero knowledge arguments and maliciously secure 2-party computation in the standard model (supporting the same function class as the underlying malicious HSS).

[*] CWI, Cryptology Group and Leiden University, pedro.capitao@cwi.nl
[†] Reichman University, hiladahari41@gmail.com
[‡] CWI, Cryptology Group, lisa.kohl@cwi.nl
[§] Xidian University, lizh0048@e.ntu.edu.sg

Contents

1 Introduction

Homomorphic secret sharing (HSS) [BGI16a] is a variant of secret sharing in which the parties are able to perform a supported class of computations locally on the shares. It can be viewed as a relaxation of somewhat or fully homomorphic encryption (FHE) to the 2- (or multi-) party setting, where the computation is distributed between non-communicating parties. HSS has proven to be a powerful tool with numerous applications, including private queries of multi-server data bases [GI14, BGI15, CBM15, BGI16b, WYG+17, BKS19], succinct preprocessing [BCGI18, BCG+17, BCG+19b], constrained pseudorandom functions [CMPR23], and more. Most noteworthy, HSS makes it possible to circumvent the circuit-size barrier of secure multi-party computation from assumptions not known to imply FHE [BGI16a, CM21, BCM23, DIJL23].

Since their introduction, numerous homomorphic secret sharing (HSS) constructions have been proposed. The groundbreaking work of [BGI16a] and follow-up results [BGI17, BCG+17] showed that based on the decisional Diffie-Hellman (DDH) assumption, it is possible to construct 2-party HSS for the class of restricted-multiplication straightline (RMS) programs.[1] Later, two-party HSS constructions for RMS programs based on the decisional composite residuosity (DCR) assumption [FGJS17, OSY21, RS21], the learning with errors (LWE) assumption (without resorting to homomorphic multiplications on ciphertexts) [BKS19, ACK23] and class groups [ADOS22] emerged, which overcame the $1/\mathsf{poly}(\lambda)$ error present in the earlier DDH-based constructions. Building on multi-key fully homomorphic encryption, multi-party HSS for all circuits can be achieved [DHRW16, COS+22]. Furthermore, a recent line of research introduced 2-party and multi-party HSS constructions for constant-degree polynomials, relying on variants of the learning parity with noise (LPN) assumption [BCG+19b, Cou19, CM21, DIJL23].

What is common to the majority of the HSS literature is that the correctness and security guarantees are in the semi-honest setting, i.e., the share generation and evaluation are assumed to be performed honestly. In fact, it is easy to see that plain HSS does not give any guarantees if the parties behave maliciously. In the setting of multi-server private information retrieval, this shortcoming has been addressed in different works on HSS with verifiable evaluation [TLM18, YO18, HZ20, CZ20, Che23, CGHJ24]. In these works the shares are assumed to be generated by an honest client, while the evaluation on the shares is performed by potentially malicious servers.

In the context of secure multi-party computation with malicious security, homomorphic secret sharing schemes have been combined with generic zero knowledge proofs [BGI16a, CM21] or message authentication codes [BEPU+22, ADOS22] to take into account malicious behavior during *share generation and evaluation*. However, all prior works either rely on random oracles (for rerandomization of the shares), or are limited to HSS for small circuit classes not capturing $\mathsf{NC}1$.

The need for rerandomization stems from the fact that most HSS techniques come with a correctness error. While some constructions [BGI16a, FGJS17, BGI17, BCG+17] involve even non-negligible correctness error that must be addressed in practical applications, all known HSS constructions for $\mathsf{NC}1$ that do not rely on somewhat or fully homomorphic encryption suffer from non-perfect correctness. This issue appears to be inherent to the current techniques, and has even been demonstrated to be so in the context of lattice-based constructions relying on the so-called rounding and lifting technique [ACK23].

This error constitutes an issue towards achieving malicious security of HSS in the standard model (even assuming honest generation of keys), as an adversary can choose its inputs and randomness adaptively in order to increase the error probability to non-negligible and thereby potentially learn something about the secret key of the other party or the underlying computation. In fact, common correctness definitions [BGI16a, BKS19, OSY21] cover only non-adaptive correctness, where correctness is guaranteed to hold only if the inputs are chosen independently of the setup parameters. This is problematic in applications such as secure computation even if relying on generic zero knowledge proofs, unless it is ensured that the parties have already committed to their inputs and randomness to be used *before* learning the HSS setup information, making the setup non-reusable across different inputs, or if additional measures are in place (e.g., if shares are re-randomized at each step of the computation using a non-predictable source of randomness such as a random oracle).

1.1 Our Contribution

In this paper, we show how to lift the non-adaptive correctness of HSS to *adaptive* correctness, which allows HSS inputs to be chosen adaptively, even after seeing the keys. To this end, we follow the derandomization techniques of Dwork et al. [DNR04] to show that any HSS satisfying non-adaptive

[1] This class of programs captures branching programs and $\mathsf{NC}1$ circuits [BGI16a].

correctness (where the correctness error can be made sufficiently small) can be transformed into an HSS satisfying adaptive correctness. On the positive side, this allows us to lift the 3-round protocol of [ADOS22] for secure 2-party computation to the standard model (preserving the round complexity).

On the negative side, this transformation comes with a significant loss: if the original HSS scheme supports evaluation of a function class $\mathcal{F}$ which takes inputs $x \in \{0,1\}^N$, then the shares of the new scheme scale with size $|\mathcal{F}'| \cdot N \cdot \mathsf{poly}(\lambda)$, where $\mathcal{F}' \subset \mathcal{F}$ is the function class supported by the new HSS. Note that this still gives succinct secure computation, as the size of the shares only depends on the size of the function class, i.e., the number of functions in $\mathcal{F}$, and not on the size of the functions $f \in \mathcal{F}'$.[2]

Our second main contribution concerns addressing the question of whether the transformation from NIZK to rate-1 NIZK via FHE [GGI+15] can be lifted to the HSS setting. Namely, we ask if an HSS allows to transform a (DV-)NIZK into a rate-1 DV-NIZK.

Towards addressing this question, we observe that many HSS schemes in literature actually have a baked-in verification procedure (e.g., [BKS19, ADOS22]) even without adding extra verification mechanisms such as message authentication codes, which we abstract as *robust linear reconstruction*. We show that this property can not only be used generically towards interactive verification of the shares (as also observed in [MORS25]), but also to safely release "reduced" shares (where the reduction depends on the correct result of the function evaluation), while still being able to verify correctness of the evaluation. We formalize these properties as *reduced share simulatability* and *verifiability*, together capturing the notion of security against malicious adversaries, and we call a scheme with these traits malicious HSS.

We further show a transformation of NIZK to compact DV-NIZK (i.e., with proof size independent of the circuit size of the statement to be proven) which, using the transformation from non-adaptive to adaptive correctness from above, can be based on any HSS supporting robust linear reconstruction.

Since, as described above, the transformation from non-adaptive correctness to adaptive correctness is not rate-preserving, we unfortunately do not get a transformation from standard HSS to rate-1 DV-NIZK, which we leave as an interesting open question.

For completeness, we further give a direct construction of *rate*-1 malicious HSS from leveled homomorphic encryption. While this does not yield any applications that are not already known to be implied directly by leveled FHE, it appears to point out a qualitative difference of SHE/FHE-based HSS schemes and other HSS techniques for $\mathsf{NC1}$ (such as "rounding and lifting" and "distributed discrete logarithm" techniques) even in a distributed setting.

In the following, we give a high-level overview of our techniques towards the transformation of NIZK to DV-NIZK via HSS with robust linear reconstruction.

1.2 Technical Overview

In this work, we focus on public-key 2-party HSS with non-interactive key aggregation. Even though HSS with non-interactive key aggregation is not formalized in prior work, we note that the HSS schemes of [BKS19, ADOS22] can be phrased as such.[3] More precisely, an HSS with non-interactive key aggregation for a function family $\mathcal{F}$ consists of a tuple of algorithms $\mathsf{HSS} = (\mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Comb})$, such that:[4]

- KeyGen on input party index b outputs a key pair $(\mathsf{pk}_b, \mathsf{sk}_b)$.
- $\mathsf{KeyComb}$ on input of public keys $\mathsf{pk}_0, \mathsf{pk}_1$, outputs a joint public key pk.
- Enc on input of a combined public key pk and an input x, outputs a ciphertext ct.
- Eval on input of a party index b, an evaluation key sk_b, a set of n ciphertexts $(\mathsf{ct}_i)_{i \in [n]}$, and a function $f \in \mathcal{F}$, computes an output share t_b.
- Comb on input of output shares t_0, t_1 returns an output value y. For now, we will consider additive reconstruction, i.e., Comb simply returns $y := t_0 + t_1$.

[2] In particular, the HSS can even support a large function class $\mathcal{F}$ as long as the class of "eligible" functions is fixed to a polynomially-large set, independent of $|f|$, at the time of setup. Once this is done, the HSS parameters can be reused for arbitrary evaluations within this class.

[3] HSS with non-interactive key aggregation has in fact be considered in [ADOS22] as HSS with one-round setup (see Fig. 5/ Thm. 2 in the EPRINT version).

[4] Note that in the following to simplify notation we omit the Setup algorithm and public parameters and only consider function families with output in $\{0,1\}$.

To satisfy correctness (with additive reconstruction), for all honestly generated parameters, all inputs x_i and all $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, x_i)$ it should hold:

$$\mathsf{Eval}(0, \mathsf{sk}_0, (\mathsf{ct}_i)_{i\in[n]}, f) + \mathsf{Eval}(1, \mathsf{sk}_1, (\mathsf{ct}_i)_{i\in[n]}, f) = f(x_1, \dots, x_n).$$

Towards security, there should exist an efficient simulator such that for any input x and any party index $b \in \{0,1\}$ it holds:

$$\{\mathsf{Sim}(\mathsf{pk}, \mathsf{sk}_b)\} \approx \{(\mathsf{Enc}(\mathsf{pk}, x), \mathsf{sk}_b)\},$$

i.e., an encryption does not reveal the underlying value even given a secret key share sk_b.

Here, one can view $(\mathsf{Enc}(\mathsf{pk}, x), \mathsf{sk}_0), (\mathsf{Enc}(\mathsf{pk}, x), \mathsf{sk}_1)$ as a 2-out-of-2 secret sharing of x.[5] Note that given a somewhat-homomorphic encryption scheme with non-interactive key aggregation one can instantiate the above homomorphic secret sharing simply by evaluating the function homomorphically on the ciphertexts and outputting the result and the secret key share.[6]

Towards malicious HSS. However, it is easy to see that this definition is not robust against malicious behavior:

- An adversarial party might generate a public key $\hat{\mathsf{pk}}_b$ maliciously.
- An adversarial party might generate an encryption $\hat{\mathsf{ct}}_i$ maliciously.
- An adversarial party might return a wrong output share $\hat{t}_b$.

The first point is relatively straightforward to address: one can simply let the adversary prove that pk_b and ct_i were indeed generated correctly using a suitable non-interactive zero knowledge proof of knowledge.

At first glance, it might seem that the same is true to ensure correctness of the encrypted ciphertext. In fact, one can ensure that the ciphertext is well-generated with the same techniques,[7] but this does not address the fact that an adversary may adaptively choose their input and randomness in such a way that evaluation might result in a wrong share with high probability. This can be prevented by letting the adversary commit to its input and randomness ahead of time. The focus of this work, however, is to construct HSS where the adversary can choose its input after seeing the setup, thereby making the HSS setup reusable for many inputs.

Finally, we have to address the possibility that an adversary could deviate from the protocol during evaluation. The first idea is again to add generic zero-knowledge proofs. While adding zero-knowledge to the key generation and ciphertext introduces an overhead that is independent of the circuit size,adding zero-knowledge to the evaluation algorithm would result in an overhead that scales with the circuit size of the function to be evaluated, which is not desirable.

HSS with robust linear reconstruction (RLR-HSS). The starting point to overcome this issue is the observation that many HSS constructions actually have a built-in mechanism which can be exploited for the purpose of verification. Namely, the output shares (t_0, t_1) of certain 2-party HSS schemes [BKS19, OSY21, RS21, ADOS22] are vectors such that $t_0 + t_1 = f(\mathbf{x}) \cdot \mathsf{sk}$. We will refer to such a scheme in the following as HSS with *robust linear reconstruction*.[8] When first introduced, this structure was an artifact of the techniques, solely used in HSS evaluation, and disregarded at the end of the computation: namely, the evaluation key is chosen such that $\mathsf{sk}[1] = 1$, and the parties output the first entry of t_0 and t_1, respectively, to recover $f(\mathbf{x})$.

Note, however, that this structure can actually be used to check if the evaluation was performed honestly (assuming honest share generation): namely, server S_b provides the full share t_b at the end of the computation, and the client (in knowledge of sk) accepts if and only if $t_0 + t_1 = f(\mathbf{x}) \cdot \mathsf{sk}$. (This has first been observed in the context of verifiable homomorphic secret sharing [Che23].) Intuitively, this allows a client to check if the evaluation has been performed honestly (assuming non-colluding servers),

[5] For this reason we sometimes refer to the process of creating keys and ciphertexts as *share generation*.

[6] The described approach does not allow for additive reconstruction. To achieve this, one can use distributed decryption and "spooky rounding" [DHRW16]).

[7] Note that here it is crucial that ct_i is tagged with a party index b to prevent that the adversarial party "copies" an honest party's input. We will omit this in the following, but will make it explicit in the formal definition.

[8] Towards malicious security, we actually require the scheme to satisfy perfect correctness (except with negligible probability over the setup algorithm). We will come back to this issue later, and for now assume that the underlying HSS schemes satisfies perfect correctness.

as a malicious server S_b would need to provide $\hat{t}_b := t_b + (-1)^{f(\mathbf{x})} \cdot \mathsf{sk}$ in order to flip the output from $f(\mathbf{x}) \cdot \mathsf{sk}$ to $(1 - f(\mathbf{x})) \cdot \mathsf{sk}$, which would imply breaking security of the HSS by recovering the secret key.

Our first observation is that this technique can be extended to a distributed setting, where no party has knowledge of the secret key sk. Namely, assume that party P_b holds sk_b such that $\mathsf{sk}_0 + \mathsf{sk}_1 = \mathsf{sk}$. Then, in case $f(\mathbf{x}) = 0$ it holds $t_0 + t_1 = 0$ and in case $f(\mathbf{x}) = 1$ it holds $(t_0 - \mathsf{sk}_0) + (t_1 - \mathsf{sk}_1) = 0$. Note here that it is no longer safe to reveal the shares $(t_b, t_b - \mathsf{sk}_b)$ in the clear, as this would allow the other party to recover sk. This can be addressed in two ways. If the output of the computation is known (as will be the case in the DV-NIZK application, where the prover wants to show $f(\mathbf{x}) = 1$), the parties can safely release only the corresponding shares $t_b - f(\mathbf{x}) \cdot \mathsf{sk}_b$. If the output of the computation is not known, as is the case in applications to secure computation, the parties can instead recover the correct output using private set intersection, where party P_b inputs $\{(-1)^b \cdot t_b, (-1)^b \cdot (t_b - \mathsf{sk}_b)\}$.[9]

Formalizing malicious security. Towards formalizing malicious security, we extend the definition of homomorphic secret sharing by relaxing the share reconstruction algorithm Comb into two algorithms Reduce and $\mathsf{RedComb}$, where Reduce additionally gets as input the (presumably) correct function output y, and security is required to hold relative to the reduced output shares. In the above example, Reduce would get as input a bit b, share t_b, secret key share sk_b, and a function output $y = f(\mathbf{x})$ and output $u_b := (y, t_b - y \cdot \mathsf{sk}_b)$. The $\mathsf{RedComb}$ algorithm on input $(y, \tilde{u}_0), (y, \tilde{u}_1)$ would output y if and only if $\tilde{u}_0 + \tilde{u}_1 = 0$. If the output space is small (e.g. $\{0,1\}$), we no longer have to consider the share combination algorithm Comb explicitly. Instead, we can define Comb as follows: compute $u_b^y \leftarrow \mathsf{Reduce}(b, \mathsf{sk}_b, t_b, y)$ for $y \in \{0, 1\}$ and return y if and only if $\mathsf{RedComb}(u_0^y, u_1^y) = y$.[10] Defining these additional algorithms Reduce and $\mathsf{RedComb}$ lets us capture the following:

1. The reduced output share u_b relative to the correct output $y = f(\mathbf{x})$ can be released without revealing anything about the underlying inputs or computation;

2. The reduced output share u_b is sufficient to verify that the computation has in fact been performed correctly.

We will formalize these two properties as *reduced share simulatability* and *verifiability*. In a nutshell, towards reduced share simulatability we require an efficient simulator Sim such that for $b \in \{0, 1\}$ it holds

$$\{\mathsf{Sim}(\mathsf{pk}_b, f, f(\mathbf{x}))\} \approx \{\mathsf{pk}_{1-b}, (\mathsf{ct}_i)_{i\in[n]}, u_{1-b}\},$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$, $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i)$, $t_b \leftarrow \mathsf{Eval}(b, \mathsf{sk}_b, f, (\mathsf{ct}_i)_{i\in[n]})$ and $u_{1-b} \leftarrow \mathsf{Reduce}(1-b, \mathsf{sk}_{1-b}, t_{1-b}, f(\mathbf{x}))$, even for adversarially generated public key $\mathsf{pk}_b \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_{1-b})$. In other words, a reduced output share can be simulated given only knowledge of the output $f(\mathbf{x})$.

The formalization of *verifiability* (where the shares are potentially generated dishonestly) turns out a bit more involved, as we have to capture the fact that:

i. If verification passes, then the computation has been performed correctly (even if the inputs and randomness are chosen maliciously based on the setup).

ii. If verification passes, it is safe to reveal the honest party's share u_{1-b} (even if this is derived from potentially maliciously generated shares).

Now, some or all of the ciphertexts are provided by the adversary and thus corresponding inputs must be extracted during simulation. One might think that due to the second point verifiability encompasses reduced share simulatability, but the two properties are actually incomparable, as in reduced share simulatability the adversary gets to see u_{1-b} *without* having to provide its own reduced share u_b to the simulator. While this definition of malicious security is somewhat cumbersome, we show in this paper that it is not necessary to instantiate it directly. Rather, we are able to give a generic transformation from any RLR-HSS to malicious HSS, where we use non-interactive zero knowledge proofs to enforce the correct generation of pk_b and ct_i and rely on robust linear reconstruction to achieve verifiability.

[9] Since in the context of secure computation, the techniques of [ADOS22] allow for non-interactive reconstruction and therefore better round complexity, we do not focus on the application of this trick to secure computation in the paper.

[10] While this definition might seem limited to HSS with small output spaces $\mathcal{Y}$ on first glance, it also captures more generic HSS: namely, if the HSS support Reduce, $\mathsf{RedComb}$, where Reduce is *independent* of the output value y, the combination algorithm above is efficient even for large spaces $\mathcal{Y}$ (as the reduce algorithm yields a single output share u_b). We refer to such HSS as HSS with non-interactive share reduction in the following.

From malicious HSS to DV-NIZK. We show that malicious HSS implies a very simple designated verifier non-interactive zero-knowledge argument, essentially lifting the construction of rate-1 NIZK from fully homomorphic encryption (FHE) [GGI+15] to HSS. Recall that in the construction of NIZK from FHE, in order to prove $f(\mathbf{w}) = 1$, the prover encrypts the witness $\mathbf{w}$ using a fully homomorphic encryption scheme and proves that the homomorphic evaluation of function f on $\mathbf{w}$ indeed results in an encryption of 1 (by opening the evaluated ciphertext). To achieve rate-1 NIZK, the paper uses hybrid encryption, i.e., encrypts the witness as $\mathbf{w} + G(s)$ for some PRG G, and then encrypts s using the FHE scheme (while also proving well-formedness of the FHE encryption). Since the verifier can check that the output ciphertext was indeed obtained via homomorphic evaluation of f on the input ciphertext and all FHE-encryptions are well-formed, soundness follows.

In this paper, we show that HSS gives rise to a similar construction resulting in DV-NIZK for languages where the evaluation of f is supported by the underlying HSS. The verifier generates $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and sends pk_1 to the prover. The prover generates its own key pair $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$, encrypts its witness $\mathbf{w}$ under the combined public key pk, evaluates f homomorphically on ct to obtain t_0, computes the reduced share u_0 relative to output $y = 1$ and sends $\pi := (\mathsf{pk}_0, \mathsf{ct}, u_0)$ to the verifier. The verifier then computes t_1 by evaluating f on ct using its own secret key share sk_1, computes its own reduced share u_1 relative to $y = 1$ and outputs $\mathsf{RedComb}(u_0, u_1)$. It can be seen that the resulting proof system satisfies zero knowledge because of the reduced share simulatability of the HSS and reusable soundness because of the verifiability of the HSS. Further, the transformation preserves the rate of the underlying HSS. Unfortunately, as we will explain below, dealing with the error in non-SHE/FHE-based HSS constructions does not result in rate-1 DV-NIZK. Yet, we make the following two observations:

- We obtain a transformation from (DV-)NIZK to *compact* (i.e., with overhead independent of the circuit size) DV-NIZK for $\mathsf{NC1}$ from any HSS for $\mathsf{NC1}$ satisfying non-adaptive robust linear reconstruction.

From malicious HSS to malicious 2PC. Finally, we give a definition of verifiability for HSS with non-interactive share reduction and show that it suffices to instantiate malicious secure 2-party computation from malicious HSS, with the property that the setup and public keys can be reused across many secure computations. The underlying HSS can be instantiated from [ADOS22]. While the MPC protocol readily follows from [ADOS22], our contribution is the generic transformation from non-adaptive to adaptive correctness (explained below), which allows to lift the result of [ADOS22] in the random oracle model to the standard model (at the cost of larger HSS parameters).

The issue of error. A crucial point we have not addressed so far is that the mentioned HSS constructions of [BKS19, OSY21, RS21] do not actually achieve perfect correctness. Instead, they settle for a non-adaptive version of correctness, where correctness is only guaranteed to hold with overwhelming probability over the random coins of KeyGen and Enc. If the adversary is able to choose the input x and/or the encryption randomness for Enc *after* seeing the evaluation key, this definition no longer gives any guarantees (even if the adversary follows the protocol honestly otherwise). This is not merely an artifact of the choice of a "too weak" definition but hints to an actual limitation of current techniques.

Namely, a multiplication $x \cdot y$ in HSS constructions can typically be viewed as a distributed decryption of a ciphertext $\mathsf{Enc}(\mathsf{pk}, x)$ with secret shares of $y \cdot \mathsf{sk}$. The issue is that this multiplication gives output shares over a different group than the one started from, so in order to continue the computation the parties have to perform a "share conversion" procedure. An example of share conversion is the lifting procedure of [BKS19], where the parties lift shares $z_0 + z_1 = z \mod p$ to additive shares of z modulo q (for a larger modulus q). It turns out that if $|z| \ll p$ and z_0, z_1 are chosen at random, then with high probability it holds $z_0 + z_1 = z$ *over the integers*, and thus also modulo q as required.[11] However, there is always a non-zero chance of an error. Even more, as observed in [ACK23], even only given access to one share, the adversary can see whether such an error potentially occurs. Namely, an error can only occur if both z_0 *and* z_1 are in the "bad area" between $-p/2 + 1$ and $-p/2 + |z|$ or both in the "bad area" between $p/2 - |z|$ and $p/2$.[12] Furthermore, the shares z_0 and z_1 are exactly $|z|$ apart modulo p, so if the adversary manages to enforce its share into the "correct" bad area (which depends on the value of z), a share conversion error will indeed occur. As the shares depend on the encryption $\mathsf{Enc}(\mathsf{pk}, x)$, an adversary can guess the value of z and choose $\mathsf{Enc}(\mathsf{pk}, x)$ such that its respective share lands in the

[11] Note that by choosing $p \geq |z| \cdot \lambda^{\omega(1)}$, the probability of a "lifting" error can be made negligible.

[12] For simplicity, here we assume that p is even and $\mathbb{Z}_p$ is represented as $\{-p/2 + 1, \ldots, p/2\}$.

"bad area".[13] By aiming for the interval borders $-p/2+1$ or $p/2$, the adversary can even increase the probability of the attack to 50% (as in this case it only needs to guess correctly whether z is a positive or negative integer).

Towards overcoming the issue of error. One idea to overcome the attack is to make use of the observation in [ACK23] to our advantage: if an honest party ever observes a share in the "bad area" during evaluation, it simply aborts. There are some issues with this approach, though. First of all, whether an abort occurs or not depends on the secret key share sk_{1-b} of the honest party, and therefore only results in an HSS with "single-key" use. In the context of DV-NIZK, this observation suffices to construct a one-time DV-NIZK, where the verifier has to generate a fresh verification key for each statement to be proven. However, for applications such as secure computation this approach is not suitable, as whether an abort occurs or not depends on the intermediary values of the underlying computation (namely, the value $z = x \cdot y$ from above). In the context of secure computation this means that an adversary can learn information about the honest party's input based on whether the honest party aborts. The approach of achieving robustness via abort is therefore not productive in this setting.

Another approach is to rely on an idealized source of randomness such as a random oracle for rerandomization. If carefully implemented, this indeed yields correctness as required even in a malicious setting. This approach has been considered in the context of secure computation with malicious security in the random oracle model by [BEPU$^+$22, ADOS22]. In this work, however, we focus on achieving HSS with malicious security in the standard model.

From negligible error to perfect correctness. We show instead that one can use techniques from Dwork et al. [DNR04] to get rid of the error, albeit unfortunately at the cost of a blow-up in parameter sizes. We start by recalling their strategy to get rid of small error in the context of public key encryption. The idea is that one can transform a public-key encryption scheme that has small correctness error (say, at most error $\epsilon = 2^{-4\lambda}$ for message space $\{0,1\}^\lambda$), into a public key encryption scheme that is perfectly correct *except on a negligible fraction of the public keys.* That is, once the public key is fixed the scheme will satisfy correctness relative to all possible messages m and encryption randomness r (except with negligible probability over the choice of the key pair). Their idea is to replace the encryption randomness $r \in \{0,1\}^{\ell(\lambda)}$ by randomness $G(s) \oplus r' \in \{0,1\}^{\ell(\lambda)}$, where $G \colon \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ is a pseudorandom generator and $r' \leftarrow_R \{0,1\}^{\ell(\lambda)}$ is a random string added to the public key. First of all, since the error probability is at most $\epsilon = 2^{-4\lambda}$, by a counting argument there must exist a fraction of at least $1-2^{-\lambda}$ "good key pairs", for which the probability of error is at most $2^{-3\lambda}$. For all "good key pairs" $(\mathsf{pk}, \mathsf{sk})$ we thus have

$$\Pr[\mathsf{Dec}(\mathsf{sk}, (\mathsf{Enc}(\mathsf{pk}, m; G(s) \oplus r')) \neq m] \leq 2^{-3\lambda}$$

where the probability is taken over $m \leftarrow_R \{0,1\}^\lambda$, $s \leftarrow_R \{0,1\}^\lambda$ and $r' \leftarrow_R \{0,1\}^{\ell(\lambda)}$. Finally, via a union bound over m and s we obtain that for all "good key pairs"

$$\Pr[\mathsf{Dec}(\mathsf{sk}, (\mathsf{Enc}(\mathsf{pk}, m; G(s) \oplus r')) \neq m] \leq 2^{-\lambda}$$

for all possible $m, s \in \{0,1\}^\lambda$, where the probability is taken over the random choice of $r' \leftarrow_R \{0,1\}^{\ell(\lambda)}$. Therefore, of the remaining good keys at most a fraction of $2^{-\lambda}$ of the keys lead to a correctness error on any message, resulting in a scheme with perfect correctness except with probability $2 \cdot 2^{-\lambda}$ over the choice of the key pair.

We can adapt this technique to HSS, where we want that correctness holds except with negligible probability over the setup parameters *for all possible key pairs* $(\mathsf{pk}, \mathsf{sk})$ *and for all possible encryptions of messages* $x_1, \ldots, x_n \in \{0,1\}^N$. To achieve this, we have to perform a union bound over $2^{2\lambda + n(N+\lambda)}$ values corresponding to the randomness of the choice of the key pairs $(\mathsf{pk}_b, \mathsf{sk}_b)$ and the randomness and message of all n ciphertexts, rather than only $2^{2\lambda}$ values as before. We therefore have to start with correctness error $\epsilon \leq 2^{-4\lambda - n(N+\lambda)}$. In HSS constructions with negligible correctness error such as [BKS19, OSY21, RS21], we can decrease the error probability by choosing larger parameters. For the HSS of [BKS19], for example, the correctness error scales at least with $\max(1/p, p/q) \cdot |f|$, where $|f|$ is the number of multiplication gates when f is written as restricted multiplication straightline program, and where p is the plaintext modulus and q the ciphertext modulus of the scheme. Thus, one can choose $p = 2^{4\lambda + n(N+\lambda)} \cdot |f|$ and $q = p^2$. On the downside, this means that even sending a single field element in $\mathbb{F}_q$ requires communication $\log q = 2 \cdot (4\lambda + n(N+\lambda)) \cdot \log |f|$. We can therefore not hope to achieve

[13] Note that enforcing z_b to land in a certain area is not trivial as the parties perform a "rounding" operation before the lifting, but the adversary can perform a similar attack on the rounding to enforce an error with high probability.

constant rate (i.e., ciphertext size $N + \mathsf{poly}(\lambda)$) via this transformation, even when using additional measures such as hybrid encryption $(m \oplus G(r), \mathsf{Enc}(\mathsf{pk}, r))$ to encrypt a message $m \in \{0,1\}^N$, as the base field $\mathbb{F}_q$ is simply too large.

From homomorphic encryption to HSS with perfect correctness. To complement the above generic transformation, we give an alternative construction of HSS that achieves perfect correctness and rate-1 ciphertexts based on leveled homomorphic encryption (LHE). We note that one cannot achieve perfectly additive reconstruction even based on LHE, as this requires "spooky rounding" [DHRW16] which comes with a non-zero error. Instead, the parties can release their shares *before rounding*, so that reconstruction can performed with perfect correctness. To allow for verification, we let the parties compute $f(x) \cdot \mathsf{sk}$ instead of $f(x)$.[14] The algorithm $\mathsf{RedComb}$ then takes inputs $(t_b, t_b - \mathsf{sk}_b)$ and outputs 0 if $t_0 \approx t_1$ and 1 if $t_0 - \mathsf{sk}_0 \approx t_1 - \mathsf{sk}_1$. To avoid the error leaking anything about the underlying noise, the parties can use noise flooding (assuming a superpolynomial ciphertext modulus). Distributed reconstruction becomes a bit more difficult in this case, as the parties have to check approximate matching in a distributed setting. One way to instantiate this is to rely on structure-aware PSI [GRS22, GRS22, vBP24]. Although we are not aware of any application of the obtained HSS that cannot be obtained directly from the underlying LHE scheme, we provide this transformation to show a qualitative difference of homomorphic encryption and HSS techniques to our current understanding even in the distributed setting.

1.3 Related Work

In this section we give a short summary of relevant works mentioned during the introduction and further related research.

Malicious secure computation from HSS. Prior works achieve malicious secure computation either by relying on generic zero knowledge proofs [BGI16a, CM21] or random oracles [BEPU+22, ADOS22]. While [CM21] achieves reusable setup (as they do not have to deal with error in the underlying HSS), their approach only yields secure computation that scales with $\log s / \log\log s$ in the circuit size s (due to their underlying HSS not supporting $\mathsf{NC1}$). The approach of [ADOS22], on the other hand, uses message authentication codes to get the round complexity down to 3 rounds (essentially, they manage to keep the reconstruction phase non-interactive). Their approach, however, relies crucially on a random oracle to rerandomize the shares. We expect that using similar derandomization techniques as presented in our work would result in a protocol for secure computation with malicious security in 3 rounds in the standard model and leave it as an interesting open question to explore this direction further.

Verifiable HSS. The line of work on verifiable HSS [TLM18, YO18, HZ20, CZ20, Che23, CGHJ24] considers a server-client setting in which the shares are generated by an honest client and malicious behavior of the server is to be detected. The work of [CGHJ24] gives an even stronger guarantee of public verifiability, i.e., verifiability is guaranteed to hold even if all servers are malicious. This setting is incomparable to ours: while we give guarantees even if the key generation is performed maliciously, we rely on at least one party behaving honestly throughout the computation.

Verifiable and malicious FSS. Much work has been dedicated to achieving verifiable or even malicious secure function secret sharing [BGI16b, BCG+19a, BCG+20, BBC+21, dCP22, Wag22, BBC+23]. This line of work crucially relies on the structure of the output for verification, and therefore is limited to FSS for simple functions such as point functions, where the parties end up with output vectors that differ at exactly one point. In the setting of HSS, however, there is typically no structure in the output itself that can be used towards verification purposes.

Multi-key HSS. Recently, [CDH+25] proposed the first multi-key HSS construction for $\mathsf{NC1}$ based on DDH, serving as the secret-sharing counterpart to multi-key FHE. This differs from HSS with non-interactive key aggregation in that in multi-key HSS the parties can generate encrypted ciphertexts *before* exchanging their shares of the public key. We expect their techniques to be compatible with our transformation, which would result in a transformation from NIZK to compact NIZK based on HSS, as

[14] We show that this can be achieved with GSW encryption [GSW13] without any additional circular security assumption.

well as malicious secure 2-round computation in the common-reference-string model. We leave a further exploration of this direction as an interesting open question.

DV-NIZK. A series of works have focused on constructing DV-NIZKs using Σ-protocols and public-key cryptographic primitives [PsV06, DFN06, VV09, CG15, Lip17, CC18]. However, as pointed out by [KNYY19b], these works either do not provide full reusability or are limited to a specific NP language. Subsequently, in a parallel line of research [CH19, KNYY19a, QRW19], non-compact reusable DV-NIZKs for all NP languages were developed based on the computational Diffie-Hellman assumption. The line of work [KNYY19b, KNYY19a, KNYY20] constructed compact and reusable DV-NIZKs based on Diffie-Hellman-like assumptions in pairing-free groups. For the special class of $\mathsf{NC1}$ circuits, they even provided rate-1 DV-NIZK. (DV-)NIZKs with statistical zero knowledge have been constructed in [LPWW20, BDS24]. Here, the latter constructs NIZKs with statistical zero knowledge and proof size $|w| + |w|^{\epsilon} \cdot \mathsf{poly}(\lambda)$ for small constant $\epsilon < 1$. Towards generic transformations, [GGI$^+$15] gave a transformation from (DV-)NIZK to rate-1 (DV-)NIZK based on FHE, whereas [BCM$^+$24] recently showed a transformation from one-time DV-NIZK to reusable DV-NIZK from a public-key pseudorandom correlation function for OT.

2 Preliminaries

For a natural number n, we denote by $[n]$ the set $\{1, \dots, n\}$. We denote by λ the security parameter. We use $\mathsf{negl}(\lambda)$ to denote any negligible function and $\mathsf{poly}(\lambda)$ for any polynomially-bounded function.

We allow the symbol $\perp$ to be used as a special output for any algorithm, which will typically indicate the presence of malicious behavior or improper inputs. As a convention, we define the output of any algorithm which receives $\perp$ as one of its inputs to be $\perp$. We assume stateful adversaries implicitly output some state at each invocation, to be used as additional input at their next invocation.

As an extension to these preliminaries, we provide definitions of some standard notions such as leveled homomorphic encryption, secure computation and (DV-)NIZK used in this paper in Appendix A.

3 Homomorphic Secret Sharing

In this section, we present our definition of homomorphic secret sharing. We note that it differs from previous works in two ways. First, we require the scheme to support non-interactive key setup. As discussed in the introduction, this does not apply to all HSS schemes, but is satisfied by natural schemes such as those from the LWE and DCR assumptions. Secondly, we split up reconstruction into algorithms Reduce and $\mathsf{RedComb}$, which can be viewed as a relaxation of standard HSS and which we elaborate on in more detail below.

Definition 3.1 (Syntax)**.** *A 2-party homomorphic secret sharing (HSS) scheme with non-interactive key aggregation for a function family* $\mathcal{F} = \{f : I^n \to \mathcal{Y}\}$ *consists of several polynomial-time algorithms* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Reduce}, \mathsf{RedComb})$*, with the following syntax:*

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*: On input the security parameter* λ*, the probabilistic algorithm* Setup *outputs a common reference string* crs*.*
- $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(b, \mathsf{crs})$*: On input party index* b *and* crs*, the probabilistic algorithm* KeyGen *outputs a public key* pk_b *and secret evaluation key* sk_b*.*
- $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$*: On input the* crs *and the distributed public key* $(\mathsf{pk}_0, \mathsf{pk}_1)$*, the deterministic algorithm* $\mathsf{KeyComb}$ *outputs a joint public key* pk*.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(b, \mathsf{crs}, \mathsf{pk}, x)$*: Given the public key* pk*, public parameter* crs*, the party index* b *and an input* $x \in I$*, the probabilistic algorithm* Enc *generates a ciphertext* ct*.*
- $t_b \leftarrow \mathsf{Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i \in [n]}, f)$*: On input of the common reference string* crs*, a public key* pk*, a party index* b*, the corresponding evaluation key* sk_b*,* n *ciphertexts* ct_i*, each with a bit* b_i *indicating which party generated the ciphertext, and a function* $f \in \mathcal{F}$*, the deterministic algorithm* Eval *computes an output share* t_b*.*
- $u_b \leftarrow \mathsf{Reduce}(b, \mathsf{sk}_b, t_b, y)$*: The probabilistic algorithm* Reduce *uses a secret key* sk_b *to convert a share* t_b *into a reduced form corresponding to an output value* $y \in \mathcal{Y}$*.*

- $y \leftarrow \mathsf{RedComb}(u_0, u_1)$*: The deterministic algorithm* $\mathsf{RedComb}$ *outputs an output value* y *or* $\perp$ *from two reduced shares* u_0*,* u_1*.*

We say that HSS *has* non-interactive share reduction*, if* $\mathsf{Reduce}(b, \mathsf{sk}_b, t_b, y)$ *is independent of* y*. In this case, we write* $\mathsf{Reduce}(b, \mathsf{sk}_b, t_b, \perp)$*.*

Note that this definition is a relaxation of the standard definition of HSS. In particular, we note that at first glance it seems to trivialize the definition to allow the Reduce algorithm to take the *correct* output y as input. Namely, to solely achieve correctness one could define $\mathsf{Reduce}(b, \mathsf{sk}_b, t_b, y)$ as $u_b := y$ (and ignore the other inputs) and let $\mathsf{RedComb}(u_0, u_1)$ return $y := u_0$. If one wants to achieve verifiability, however, this does not suffice: in fact, in this work we show how to use Eval to verify that the output was indeed computed directly.

Further, if the output space $\mathcal{Y}$ is polynomial-sized (e.g., $\mathcal{Y} = \{0, 1\}$), and the HSS scheme satisfies the notion of verifiability defined below, one can derive a reconstruction algorithm which takes as input solely the shares t_0 and t_1 and returns y from Reduce and $\mathsf{RedComb}$ as follows: for all possible outputs $\mu \in \mathcal{Y}$, compute $u_b^\mu \leftarrow \mathsf{Reduce}(b, \mathsf{sk}_b, t_b, \mu)$ for $b \in \{0, 1\}$; if $\mathsf{RedComb}(u_0^\mu, u_1^\mu) = \mu$ for some $\mu \in \mathcal{Y}$ output $y := \mu$, else output $\perp$. While this requires interaction if executed within a larger protocol (e.g., a secure computation), in our instantiations it can be implemented by a simple PSI protocol (and, in particular, with communication that is independent of the circuit size of the function f to be evaluated).

Finally, note that if the scheme supports a Reduce algorithm that does not require taking y as input, then the resulting scheme supports a standard reconstruction algorithm even for large output space. We leave it as interesting open direction to construct malicious HSS with large output space.

The following definition asserts that, in an honest execution, the reconstruction algorithm recovers the output of the function being homomorphically evaluated. This is notably stronger than the usual (non-adaptive) definition of correctness, since we require the condition to hold with overwhelming probability over the randomness of the CRS generation, for all possible inputs and random coins of the other algorithms. In contrast, previous works (such as [BKS19, OSY21]) require only correctness to hold for any input with overwhelming probability over the random coins of all algorithms involved.

Definition 3.2 (Correctness)**.** *We say* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Reduce}, \mathsf{RedComb})$ *satisfies* correctness *relative to a function family* $\mathcal{F} = \{f \colon I^n \to \mathcal{Y}\}$ *if there exists a negligible function* $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *such that for* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ *the following holds, except with probability* $\mathsf{negl}(\lambda)$ *(taken over the random coins of* Setup*). For all inputs* $x_1, \ldots, x_n \in I$ *and any function* $f \in \mathcal{F}$*, for* $b \in \{0, 1\}$*, for all* $(\mathsf{pk}_b, \mathsf{sk}_b)$ *in the image of* $\mathsf{KeyGen}(b, \mathsf{crs})$*,* $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$*, for all* ct_i *in the image of* $\mathsf{Enc}(b, \mathsf{crs}, \mathsf{pk}, x_i)$ *for* $i \in [n]$*, for all* $t_b \leftarrow \mathsf{Eval}(b, \mathsf{sk}_b, (\mathsf{ct}_i)_i, f)$*, and for all* u_b *in the image of* $\mathsf{Reduce}(b, \mathsf{sk}_b, t_b, f(x_1, \ldots, x_n))$*, it holds*

$$\mathsf{RedComb}(u_0, u_1) = f(x_1, \ldots, x_n).$$

The next definition describes a very useful property of some HSS schemes, which will be necessary for our main construction in the next section. In Section 5 we will look into how this property can be attained.

Definition 3.3 (HSS with Robust Linear Reconstruction)**.** *We say* HSS *satisfies* $\mathsf{robust\ linear\ reconstruction}$ *relative to a function family* $\mathcal{F}$ *if there exists a negligible function* $\epsilon \colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *(called the* correctness error*) such that for* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ *the following holds, except with probability* $\epsilon(\lambda)$ *(taken over the random coins of* Setup*). For all inputs* $x_1, \ldots, x_n \in I$ *and any function* $f \in \mathcal{F}$*, for* $b \in \{0, 1\}$*, for all* $(\mathsf{pk}_b, \mathsf{sk}_b)$ *in the image of* $\mathsf{KeyGen}(b, \mathsf{crs})$*,* $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$*, and for all* ct_i *in the image of* $\mathsf{Enc}(b, \mathsf{crs}, \mathsf{pk}, x_i)$ *for* $i \in [n]$*:*

- *the first component* $\mathsf{sk}[1]$ *of the vector* $\mathsf{sk} := \mathsf{sk}_0 + \mathsf{sk}_1 \in \mathbb{Z}_q^k$ *is 1;*
- $\mathsf{Eval}(0, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_0, (\mathsf{ct}_i)_i, f) + \mathsf{Eval}(1, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_1, (\mathsf{ct}_i)_i, f) = f(x_1, \ldots, x_n) \cdot \mathsf{sk}$.

We say that an HSS satisfies weakly robust linear reconstruction *relative to a function family* $\mathcal{F}$*, if the above holds with*

- $\mathsf{Eval}(0, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_0, (\mathsf{ct}_i)_i, f) + \mathsf{Eval}(1, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_1, (\mathsf{ct}_i)_i, f) = f(x_1, \ldots, x_n)$.

Observe that robust linear reconstruction implies correctness for the combination algorithm defined in the following way: $\mathsf{Reduce}(b, \mathsf{sk}_b, t_b, y)$ simply outputs $u_b := t_b$ and $\mathsf{RedComb}(u_0, u_1)$ outputs the first component of the vector $u_0 + u_1 = t_0 + t_1 = f(x_1, \ldots, x_n) \cdot (\mathsf{sk}_0 + \mathsf{sk}_1) = f(x_1, \ldots, x_n)$. We will later

$\underline{\mathsf{Real}^{\mathsf{rss}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda)\text{:}}$ $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ $(\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$ $\beta \leftarrow \mathcal{A}^{\mathcal{O}^0_{\mathsf{share}}}(\mathsf{crs}, \mathsf{pk}_{1-b})$ **return** β $\underline{\mathcal{O}^0_{\mathsf{share}}(\mathsf{pk}_b, (b_i, x_i)_{i\in[n]}, f)\text{:}}$ $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$ $\mathsf{ct}_i \leftarrow \mathsf{Enc}(b_i, \mathsf{crs}, \mathsf{pk}, x_i)\ \forall i \in [n]$ $t_{1-b} \leftarrow \mathsf{Eval}(1-b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_{1-b}, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$ $u_{1-b} \leftarrow \mathsf{Reduce}(1-b, \mathsf{sk}_{1-b}, t_{1-b}, f(x_1, \ldots, x_n))$ **output** $((\mathsf{ct}_i)_{i\in[n]}, u_{1-b})$	$\underline{\mathsf{Ideal}^{\mathsf{rss}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda)\text{:}}$ $(\mathsf{td}, \widehat{\mathsf{crs}}, \widehat{\mathsf{pk}}_{1-b}) \leftarrow \mathsf{Sim}_1(1^\lambda)$ $\beta \leftarrow \mathcal{A}^{\mathcal{O}^1_{\mathsf{share}}}(\widehat{\mathsf{crs}}, \widehat{\mathsf{pk}}_{1-b})$ **return** β $\underline{\mathcal{O}^1_{\mathsf{share}}(\mathsf{pk}_b, (b_i, x_i)_{i\in[n]}, f)\text{:}}$ $((\mathsf{ct}_i)_{i\in[n]}, \hat{u}_{1-b}) \leftarrow$ $\quad \mathsf{Sim}_2(\mathsf{td}, \mathsf{pk}_b, (b_i)_{i\in[n]}, f, f(x_1, \ldots, x_n))$ **output** $((\mathsf{ct}_i)_{i\in[n]}, \hat{u}_{1-b})$

Figure 1: HSS Reduced share simulatability.

show that we can further use the (computational) entropy of sk to verify that the result has indeed been computed correctly.

We move on to the security notions for HSS. First we recall the traditional definition, which states that HSS encryption is semantically secure in the presence of an adversary who has access to the secret key of one (but not both) of the parties. In our setting it can be formulated as follows.

Definition 3.4 (Security)**.** *We say* HSS *is* secure*, if for any (stateful) PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{>0}$ *such that for any* $\lambda \in \mathbb{N}$*, any* $b \in \{0,1\}$*, for* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}) \leftarrow \mathsf{KeyGen}(1-b, \mathsf{crs})$*,* $(\mathsf{pk}_b, x_0, x_1) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_{1-b})$ *with* $|x_0| = |x_1|$ *and* $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$ *with* $\mathsf{pk} \neq \perp$ *it holds*

$$|\Pr[\mathcal{A}(\mathsf{Enc}(1-b, \mathsf{crs}, \mathsf{pk}, x_0)) = 1] - \Pr[\mathcal{A}(\mathsf{Enc}(1-b, \mathsf{crs}, \mathsf{pk}, x_1)) = 1]| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of Setup*,* KeyGen*,* Enc *and* $\mathcal{A}$*.*

Next we define what we consider to be a maliciously secure HSS scheme. This notion requires the algorithms Reduce and $\mathsf{RedComb}$ and is split into two parts: *reduced share simulatability* and *verifiability*. In the first definition, the adversary is only given control over the key generation of one of the parties and we must be able to simulate a set of ciphertexts and a reduced share from the other party. This property will directly yield zero-knowledge for our DV-NIZK construction in Section 6. In the security game of the second definition, the adversary is allowed to freely create ciphertexts and reduced shares for the party it controls and can ask to see ciphertexts from the honest party. The adversary can then use a combination of these on a verification query. The real-world verification oracle returns a reduced share from the honest party only if verification succeeds; in the ideal world, the value of the evaluated function f must be correct and the reduced share is simulated (without knowledge of the honest party's secret key).

We call *malicious HSS* any scheme which satisfies the security properties of both Definitions 3.5 and 3.6, in addition to correctness (Definition 3.2).

Definition 3.5 (Reduced share simulatability)**.** *We say an HSS scheme* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Reduce}, \mathsf{RedComb})$ *satisfies* reduced share simulatability *relative to function class* $\mathcal{F}$*, if for all* $b \in \{0,1\}$ *there exists a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that for all PPT adversaries* $\mathcal{A}$

$$\left|\Pr\left[\mathsf{Real}^{\mathsf{rss}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}^{\mathsf{rss}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the experiments are as defined in Figure 1 and where the probability is taken over the random coins of Setup*,* KeyGen*,* Enc*,* Reduce*,* Sim *and* $\mathcal{A}$*.*

Definition 3.6 (Verifiability)**.** *We say an HSS scheme* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Reduce}, \mathsf{RedComb})$ *satisfies* verifiability *relative to function class* $\mathcal{F}$*, if for all* $b \in \{0,1\}$ *there exists a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$ *such that for all PPT adversaries* $\mathcal{A}$

$$\left|\Pr\left[\mathsf{Real}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the experiments are as defined in Figure 2 and where the probability is taken over the random coins of Setup*,* KeyGen*,* Enc*,* Reduce*,* Sim *and* $\mathcal{A}$*.*

We will require a slightly modified version of the verifiability definition for our application to secure 2-party computation. This variant is included in Appendix A.6.

$\underline{\mathsf{Real}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda)\text{:}}$
$Q := \emptyset$
$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$
$\beta \leftarrow \mathcal{A}^{\mathcal{O}^0_{\mathsf{ct}},\mathcal{O}^0_{\mathsf{ver}}}(\mathsf{crs}, \mathsf{pk}_{1-b})$
return β

$\underline{\mathcal{O}^0_{\mathsf{ct}}(\mathsf{pk}_b, x)\text{:}}$
$\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$
$\mathsf{ct} \leftarrow \mathsf{Enc}(1-b, \mathsf{crs}, \mathsf{pk}, x)$
$Q := Q \cup \{(\mathsf{pk}_b, \mathsf{ct})\}$
return ct

$\underline{\mathcal{O}^0_{\mathsf{ver}}(\mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f, y, u_b)\text{:}}$
$H := \{i \in [n] \mid b_i = 1-b\}$
if $\exists i \in H\colon (\mathsf{pk}_b, \mathsf{ct}_i) \notin Q$ **output** $\perp$
else:
  $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$
  $t_{1-b} \leftarrow \mathsf{Eval}(1-b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_{1-b}, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$
  $u_{1-b} \leftarrow \mathsf{Reduce}(1-b, \mathsf{sk}_{1-b}, t_{1-b}, y)$
  if $\mathsf{RedComb}(u_0, u_1) \neq y$ **output** $\perp$
  else output u_{1-b}

$\underline{\mathsf{Ideal}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda)\text{:}}$
$Q := \emptyset$
$(\mathsf{td}, \widehat{\mathsf{crs}}, \widehat{\mathsf{pk}}_{1-b}) \leftarrow \mathsf{Sim}_1(1^\lambda)$
$\beta \leftarrow \mathcal{A}^{\mathcal{O}^1_{\mathsf{ct}},\mathcal{O}^1_{\mathsf{ver}}}(\widehat{\mathsf{crs}}, \widehat{\mathsf{pk}}_{1-b})$
return β

$\underline{\mathcal{O}^1_{\mathsf{ct}}(\mathsf{pk}_b, x)\text{:}}$
$\mathsf{ct} \leftarrow \mathsf{Sim}_2(\widehat{\mathsf{crs}}, \widehat{\mathsf{pk}}_{1-b}, \mathsf{pk}_b)$
$Q := Q \cup \{(\mathsf{pk}_b, \mathsf{ct}, x)\}$
return ct

$\underline{\mathcal{O}^1_{\mathsf{ver}}(\mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f, y, u_b)\text{:}}$
$H := \{i \in [n] \mid b_i = 1-b\}$
if $\exists i \in H\colon (\mathsf{pk}_b, \mathsf{ct}_i, \cdot) \notin Q$ **output** $\perp$
else:
  for all $i \in H$:
    $x_i := x$ **for** $(\mathsf{pk}_b, \mathsf{ct}_i, x) \in Q$
  for all $i \in [n] \setminus H$:
    $x_i \leftarrow \mathsf{Sim}_3(\mathsf{td}, \mathsf{pk}_b, \mathsf{ct}_i)$
  $\hat{y} := f(x_1, \ldots, x_n)$
  $\hat{u}_{1-b} \leftarrow \mathsf{Sim}_4(\mathsf{td}, \mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f, \hat{y}, u_b)$
  if $\hat{y} \neq y$ **output** $\perp$
  else output $\hat{u}_{1-b}$

Figure 2: HSS verifiability.

4 Constructing Malicious HSS

We show how we can obtain an HSS scheme satisfying Definitions 3.5 and 3.6 from an HSS satisfying robust linear reconstruction (Definition 3.3), together with a NIZK proof of knowledge system and a pseudorandom generator (PRG).[15]

Construction 4.1. *Let $\lambda \in \mathbb{N}$ denote a security parameter. Let $G : \{0,1\}^\lambda \to \{0,1\}^N$ be a* PRG*, with depth d_G as a boolean circuit. Let* $\overline{\mathsf{HSS}}$ *be a robust linear HSS scheme, supporting the evaluation of functions computable by polynomial-sized programs of depth up to d'. Let* NIZK *and* NIZK$'$ *be non-interactive zero-knowledge proof of knowledge systems for the language families*

$$L_{b,\overline{\mathsf{crs}}} = \{\overline{\mathsf{pk}}_b \mid \exists(\overline{\mathsf{sk}}_b, r) \text{ such that } (\overline{\mathsf{pk}}_b, \overline{\mathsf{sk}}_b) = \overline{\mathsf{HSS}}.\mathsf{KeyGen}(b, \overline{\mathsf{crs}}; r)\},$$
$$L'_{\overline{\mathsf{crs}}} = \{(\overline{\mathsf{ct}}, \overline{\mathsf{pk}}) \mid \exists(s, r) \text{ such that } s \in \{0,1\}^\lambda \text{ and } \overline{\mathsf{ct}} = \overline{\mathsf{HSS}}.\mathsf{Enc}(\overline{\mathsf{pk}}, s; r)\},$$

respectively. Then, the following construction is an HSS supporting the evaluation of functions f computable by polynomial-sized programs of depth up to $d = d' - d_G$, with one or more inputs in $\{0,1\}^N$ and output in $\{0,1\}$.

- $\mathsf{crs} \leftarrow \mathsf{HSS}.\mathsf{Setup}(1^\lambda)$*: Generate* $\overline{\mathsf{crs}} \leftarrow \overline{\mathsf{HSS}}.\mathsf{Setup}(1^\lambda)$ *and, for $b = 0, 1$,* $\mathsf{crs}^b_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK}.\mathsf{Setup}(\overline{\mathsf{crs}})$, $\mathsf{crs}^b_{\mathsf{NIZK}'} \leftarrow \mathsf{NIZK}'.\mathsf{Setup}(\overline{\mathsf{crs}})$. *Output* $\mathsf{crs} := (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$.
- $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{HSS}.\mathsf{KeyGen}(b, \mathsf{crs})$*: Parse* $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. *Sample* $(\overline{\mathsf{pk}}_b, \overline{\mathsf{sk}}_b) \leftarrow \overline{\mathsf{HSS}}.\mathsf{KeyGen}(b, \overline{\mathsf{crs}}; r)$ *for random r and generate* $\pi_b \leftarrow \mathsf{NIZK}.\mathsf{Prove}(\mathsf{crs}^b_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_b, (\overline{\mathsf{sk}}_b, r))$. *Set* $\mathsf{pk}_b := (\overline{\mathsf{pk}}_b, \pi_b)$ *and* $\mathsf{sk}_b := \overline{\mathsf{sk}}_b$ *and output* $(\mathsf{pk}_b, \mathsf{sk}_b)$.
- $\mathsf{pk} \leftarrow \mathsf{HSS}.\mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$*: Parse* $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. *For $b \in \{0,1\}$ parse* $\mathsf{pk} = (\overline{\mathsf{pk}}_b, \pi_b)$ *and let* $\overline{\mathsf{pk}} \leftarrow \overline{\mathsf{HSS}}.\mathsf{KeyComb}(\overline{\mathsf{crs}}, \overline{\mathsf{pk}}_0, \overline{\mathsf{pk}}_1)$. *If* $\mathsf{NIZK}.\mathsf{Verify}(\mathsf{crs}^0_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_0, \pi_0) = \mathsf{NIZK}.\mathsf{Verify}(\mathsf{crs}^1_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_1, \pi_1) = 1$*, output* $\mathsf{pk} := \overline{\mathsf{pk}}$*; otherwise output* $\perp$.

[15] The use of a PRG in this construction is only necessary if we are interested in obtaining compactness; otherwise, it can be simplified by doing away with the "hybrid encryption" technique.

- $\mathsf{ct} \leftarrow \mathsf{HSS.Enc}(b, \mathsf{crs}, \mathsf{pk}, x)$: *For* $x \in \{0,1\}^N$ *proceed as follows. Parse* $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. *Generate a seed* $s \leftarrow_R \{0,1\}^\lambda$, *an HSS encryption* $\overline{\mathsf{ct}} \leftarrow \overline{\mathsf{HSS}}.\mathsf{Enc}(\mathsf{pk}, s; r)$, *and a proof* $\pi \leftarrow \mathsf{NIZK}'.\mathsf{Prove}(\mathsf{crs}^b_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}, \mathsf{pk}), (s, r))$. *Set a one-time pad encryption* $\mathsf{OTP} := G(s) \oplus x$. *Output* $\mathsf{ct} = (\overline{\mathsf{ct}}, \pi, \mathsf{OTP})$.

- $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$: *Parse* $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. *Parse* $\mathsf{ct}_i = (\overline{\mathsf{ct}}_i, \pi_i, \mathsf{OTP}_i)$. *Compute* $t_b \leftarrow \overline{\mathsf{HSS}}.\mathsf{Eval}(b, \overline{\mathsf{crs}}, \overline{\mathsf{pk}}, \overline{\mathsf{sk}}_b, (\overline{\mathsf{ct}}_i)_{i\in[n]}, f')$, *where*

$$f'(s_1, \dots, s_n) := f(G(s_1) \oplus \mathsf{OTP}_1, \dots, G(s_n) \oplus \mathsf{OTP}_n).$$

  If $\mathsf{NIZK}'.\mathsf{Verify}(\mathsf{crs}^{b_i}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}_i, \mathsf{pk}), \pi_i) = 1$ *for all* $i \in [n]$, *output* t_b; *otherwise output* $\perp$.

- $u_b \leftarrow \mathsf{HSS.Reduce}(b, \mathsf{sk}_b, t_b, y)$: *Output* $u_b := (y, t_b - y \cdot \mathsf{sk}_b)$.

- $y \leftarrow \mathsf{HSS.RedComb}(u_0, u_1)$: *Parse* $u_b =: (\tilde{y}_b, \tilde{u}_b)$. *If* $y_0 = y_1$ *and* $\tilde{u}_0 + \tilde{u}_1 = 0$, *output* $y := y_0$; *otherwise output* $\perp$.

Correctness. It can easily be checked that this scheme satisfies correctness, as per Definition 3.2. In fact, letting $y := f'(s_1, \dots, s_n) = f(x_1, \dots, x_n)$, for honestly generated shares $u_b := (y, \tilde{u}_b)$ we have $\tilde{u}_b = t_b - y \cdot \mathsf{sk}_b$ for $b \in \{0,1\}$ and, by the robust linear reconstruction of $\overline{\mathsf{HSS}}$, $t_0 + t_1 = y \cdot (\mathsf{sk}_0 + \mathsf{sk}_1)$. It follows that $\tilde{u}_0 + \tilde{u}_1 = t_0 - y \cdot \mathsf{sk}_0 + t_1 - y \cdot \mathsf{sk}_1 = t_0 + t_1 - y \cdot (\mathsf{sk}_0 + \mathsf{sk}_1) = 0$. Note also that all the intermediate proofs are accepted, by the completeness of NIZK and NIZK'.

Ciphertext size. HSS has rate-1 ciphertexts, as the size of an encryption of an input x is $|\mathsf{ct}| = |x| + \mathsf{poly}(\lambda)$, independently of the function to be evaluated.

Theorem 4.2 (Malicious HSS security). *Let* $\lambda \in \mathbb{N}$ *denote a security parameter. Let* $G : \{0,1\}^\lambda \to \{0,1\}^N$ *be a* PRG, *with depth* d_G *as a boolean circuit. Let* $\overline{\mathsf{HSS}}$ *be a robust linear HSS scheme, supporting the evaluation of functions computable by polynomial-sized programs of depth up to* d'. *Let* NIZK *and* NIZK' *be non-interactive zero-knowledge proof of knowledge systems for the families of languages* $L_{b,\overline{\mathsf{crs}}}$ *and* $L'_{\overline{\mathsf{crs}}}$, *respectively, as in Construction 4.1.*

Then HSS, *defined in Construction 4.1, satisfies reduced share simulatability and verifiability, as per Definitions 3.5 and 3.6, while supporting the evaluation of functions* f *computable by polynomial-sized programs of depth up to* $d = d' - d_G$, *with one or more inputs in* $\{0,1\}^N$ *and output in* $\{0,1\}$.

Proof. We define the simulators required by the security definition of reduced share simulatability.

– $\mathsf{Sim}_1(1^\lambda)$:

  1. Generate $\overline{\mathsf{crs}} \leftarrow \overline{\mathsf{HSS}}.\mathsf{Setup}(1^\lambda)$
  2. $(\mathsf{crs}^\beta_{\mathsf{NIZK}}, \mathsf{td}^\beta_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TSetup}(\overline{\mathsf{crs}})$, $\beta \in \{0,1\}$
  3. $(\mathsf{crs}^\beta_{\mathsf{NIZK}'}, \mathsf{td}^\beta_{\mathsf{NIZK}'}) \leftarrow \mathsf{NIZK}'.\mathsf{TSetup}(\overline{\mathsf{crs}})$, $\beta \in \{0,1\}$
  4. $\mathsf{crs} := (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$, $\mathsf{td} := (\mathsf{crs}, \mathsf{td}^0_{\mathsf{NIZK}}, \mathsf{td}^1_{\mathsf{NIZK}}, \mathsf{td}^0_{\mathsf{NIZK}'}, \mathsf{td}^1_{\mathsf{NIZK}'})$
  5. $(\overline{\mathsf{pk}}_{1-b}, \overline{\mathsf{sk}}_{1-b}) \leftarrow \overline{\mathsf{HSS}}.\mathsf{KeyGen}(\overline{\mathsf{crs}})$
  6. $\pi_{1-b} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$
  7. $\mathsf{pk}_{1-b} := (\overline{\mathsf{pk}}_{1-b}, \pi_{1-b})$
  8. Output $(\mathsf{td}, \mathsf{crs}, \mathsf{pk}_{1-b})$.

– $\mathsf{Sim}_2(\mathsf{td}, \mathsf{pk}_b, (b_i)_{i\in[n]}, f, y)$:

  1. Parse $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$, $\mathsf{td} = (\mathsf{crs}, \mathsf{td}^0_{\mathsf{NIZK}}, \mathsf{td}^1_{\mathsf{NIZK}}, \mathsf{td}^0_{\mathsf{NIZK}'}, \mathsf{td}^1_{\mathsf{NIZK}'})$
  2. Parse $\mathsf{pk}_b = (\overline{\mathsf{pk}}_b, \pi_b)$
  3. $(\mathsf{sk}_b, r) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}^b_{\mathsf{NIZK}}, \mathsf{td}^b_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_b, \pi_b)$
  4. $\mathsf{pk} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{crs}, \mathsf{pk}_{1-b}, \mathsf{pk}_b)$
  5. For $i \in [n]$:
     i. $\overline{\mathsf{ct}}_i \leftarrow \overline{\mathsf{HSS}}.\mathsf{Enc}(\mathsf{pk}, 0^\lambda)$
     ii. $\pi_i \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{b_i}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}_i, \mathsf{pk}))$

iii. $\mathsf{ct}_i := (\overline{\mathsf{ct}}_i, \pi_i, \mathsf{OTP}_i)$, where $\mathsf{OTP}_i \leftarrow \{0,1\}^N$

6. $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$
7. $u_{1-b} := (y, y \cdot \mathsf{sk}_b - t_b)$
8. Output $((\mathsf{ct}_i)_{i\in[n]}, u_{1-b})$.

Now we define a sequence of indistinguishable hybrid games.

- $\mathbf{H}_0$: This is the real game described in Definition 3.5, with output as generated by $\beta \leftarrow \mathcal{A}^{\mathcal{O}^0_{\mathsf{share}}(\cdot,\cdot,\cdot)}(\mathsf{crs}, \mathsf{pk}_{1-b})$, where $\mathsf{crs} \leftarrow \mathsf{HSS.Setup}(1^\lambda)$ and $(\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}) \leftarrow \mathsf{HSS.KeyGen}(\mathsf{crs})$.

- $\mathbf{H}_1$: In this hybrid we generate the crs as in Sim_1, with $(\mathsf{crs}^\beta_{\mathsf{NIZK}}, \mathsf{td}^\beta_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TSetup}(\overline{\mathsf{crs}})$ and $(\mathsf{crs}^\beta_{\mathsf{NIZK}'}, \mathsf{td}^\beta_{\mathsf{NIZK}'}) \leftarrow \mathsf{NIZK}'.\mathsf{TSetup}(\overline{\mathsf{crs}})$ for $\beta \in \{0,1\}$. Additionally, we include a simulated proof $\pi_{1-b} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$ in the public key pk, and in the oracle $\mathcal{O}_{\mathsf{share}}(\cdot,\cdot,\cdot)$ we change the proof in the ciphertexts $\mathsf{ct}_i = (\overline{\mathsf{ct}}_i, \pi_i, \mathsf{OTP}_i)$ to be $\pi_i \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{b_i}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}_i, \mathsf{pk}))$.

- $\mathbf{H}_2$: We now change the way in which the reduced share is computed, in the oracle $\mathcal{O}^0_{\mathsf{share}}(\cdot,\cdot,\cdot)$: it is now $u_{1-b} := (y, y \cdot \mathsf{sk}_b - t_b)$, where $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$, $(\mathsf{sk}_b, r) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}^b_{\mathsf{NIZK}}, \mathsf{td}^b_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_b, \pi_b)$ and $y = f(x_1, \ldots, x_n)$.

- For every $0 \le q \le q_{\mathsf{max}}$ and $0 \le i \le n_{\mathsf{max}}$, we define $\mathbf{H}_3^{q,i}$ (where q_{max} and n_{max} are upper bounds on the number of queries to $\mathcal{O}_{\mathsf{share}}$ and the number of inputs x_i in a query, respectively): In this hybrid, all ciphertexts in the first $q-1$ queries and the first i ciphertexts in the q-th query to $\mathcal{O}_{\mathsf{share}}$ are changed. Specifically, they are of the form $c_j = (\bar{c}_j, \pi_j, \mathsf{OTP}_j)$, where we change the components $\bar{c}_j$ and OTP_j to $\bar{c}_j \leftarrow \overline{\mathsf{HSS}}.\mathsf{Enc}(\overline{\mathsf{pk}}, 0^\lambda)$ and $\mathsf{OTP}_j \leftarrow \{0,1\}^N$. We remark that $\mathbf{H}_3^{0,0}$ is identical to $\mathbf{H}_2$ and $\mathbf{H}_3^{q_{\mathsf{max}},n_{\mathsf{max}}}$ is the ideal-world game of Definition 3.5.

We show that each hybrid is computationally indistinguishable from the next.

$\mathbf{H}_0 \approx \mathbf{H}_1$: By the mode-indistinguishability property of NIZK and NIZK', the two methods of creating the crs are indistinguishable. Moreover, $\pi_{1-b} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$ is indistinguishable from $\pi_{1-b} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b}, (\overline{\mathsf{sk}}_{1-b}, r))$ and $\pi_i \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{b_i}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}_i, \mathsf{pk}))$ is indistinguishable from $\pi_i \leftarrow \mathsf{NIZK}'.\mathsf{Prove}(\mathsf{crs}^{b_i}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}_i, \mathsf{pk}), (s_i, r_i))$.

$\mathbf{H}_1 \approx \mathbf{H}_2$: Consider a query $(\mathsf{pk}_b, (b_i, x_i)_{i\in[n]}, f)$ to $\mathcal{O}_{\mathsf{share}}$ and let $y = f(x_1, \ldots, x_n)$. By knowledge soundness of NIZK, the extracted key sk_b is valid: $(\overline{\mathsf{pk}}_b, \mathsf{sk}_b) = \overline{\mathsf{HSS}}.\mathsf{KeyGen}(\mathsf{pp}; r)$ for some randomness r. Therefore, by the robust linear reconstruction property of $\overline{\mathsf{HSS}}$, we have

$$t_b + t_{1-b} = f'(s_1, \ldots, s_n) \cdot (\mathsf{sk}_b + \mathsf{sk}_{1-b}).$$

Since $f'(s_1, \ldots, s_n) = y$, we can rewrite this as

$$t_{1-b} - y \cdot \mathsf{sk}_{1-b} = y \cdot \mathsf{sk}_b - t_b,$$

which shows that $\tilde{u}_{1-b} := t_{1-b} - y \cdot \mathsf{sk}_{1-b}$ (as defined in $\mathbf{H}_1$) and $\tilde{u}_{1-b} := y \cdot \mathsf{sk}_b - t_b$ (as defined in $\mathbf{H}_2$) are identical.

$\mathbf{H}_3^{q,i-1} \approx \mathbf{H}_3^{q,i}$ for all $0 \le q \le q_{\mathsf{max}}$, $0 < i \le n_{\mathsf{max}}$: We introduce an intermediate hybrid $\mathbf{H}'$ in which (for the q-th query) the i-th ciphertext $\mathsf{ct}_i = (\overline{\mathsf{ct}}_i, \pi_i, \mathsf{OTP}_i)$ has its first component changed to $\overline{\mathsf{ct}}_i \leftarrow \overline{\mathsf{HSS}}.\mathsf{Enc}(\overline{\mathsf{pk}}, 0^\lambda)$, this being its only difference from $\mathbf{H}_3^{q,i-1}$, but the last component remains as $\mathsf{OTP}_i = G(s_i) \oplus x_i$ (its only difference from $\mathbf{H}_3^{q,i}$).

We first show that $\mathbf{H}_3^{q,i-1}$ and $\mathbf{H}'$ are indistinguishable through a reduction to the security of $\overline{\mathsf{HSS}}$. Consider a PPT adversary $\mathcal{B}_{q,i}$ which, on input $(\overline{\mathsf{crs}}, \overline{\mathsf{pk}}_{1-b})$, generates $(\mathsf{crs}^\beta_{\mathsf{NIZK}}, \mathsf{td}^\beta_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TSetup}(\overline{\mathsf{crs}})$ as well as $(\mathsf{crs}^\beta_{\mathsf{NIZK}'}, \mathsf{td}^\beta_{\mathsf{NIZK}'}) \leftarrow \mathsf{NIZK}'.\mathsf{TSetup}(\overline{\mathsf{crs}})$ for $\beta \in \{0,1\}$ and sets $\mathsf{crs} := (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. Then $\mathcal{B}_{q,i}$ lets $\mathsf{pk}_{1-b} = (\overline{\mathsf{pk}}_{1-b}, \pi_{1-b})$, where $\pi_{1-b} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$, and runs $\beta \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_{1-b})$. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{share}}$ on some input $(\mathsf{pk}_b, (b_i, x_i)_{i\in[n]}, f)$, $\mathcal{B}_{q,i}$ returns $((c_j)_{j\in[n]}, u_{1-b})$, where u_{1-b} is computed as in $\mathbf{H}_2$ and the ciphertexts $c_j = (\bar{c}_j, \pi_j, \mathsf{OTP}_j)$ are generated as follows.

– If fewer than q queries have been made, or if it is the q-th query and $j < i$, then $\bar{c}_j \leftarrow \mathsf{HSS.Enc}(\overline{\mathsf{pk}}, 0^\lambda)$; if it is the q-th query and $j = i$, then $\bar{c}_j$ is obtained from the $\overline{\mathsf{HSS}}$ challenger when $\mathcal{B}$ sends the public key and pair of messages $(\overline{\mathsf{pk}}_b, s_i, 0^\lambda)$; otherwise $\bar{c}_j \leftarrow \mathsf{HSS.Enc}(\overline{\mathsf{pk}}, s_j)$.

– For all queries, $\pi_j \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{b_j}_{\mathsf{NIZK}'}, (\bar{c}_j, \mathsf{pk}))$ for all j.

– If fewer than q queries have been made, or if it is the q-th query and $j < i$, then $\mathsf{OTP}_j \leftarrow \{0,1\}^N$; otherwise $\mathsf{OTP}_j := G(s_j) \oplus x_j$.

We define the output of $\mathcal{B}_{q,i}$ as the output of $\mathcal{A}$. Since the view of $\mathcal{A}$ corresponds to $\mathbf{H}_3^{q,i-1}$ if ct_i is an encryption of s_i and to $\mathbf{H}'$ if ct_i is an encryption of 0^λ, the distinguishing advantage of $\mathcal{B}$ is identical to that of $\mathcal{A}$.

We now show that $\mathbf{H}'$ and $\mathbf{H}_3^{q,i}$ are indistinguishable through a reduction to the pseudorandomness of G. We define a PPT adversary $\hat{\mathcal{B}}_{q,i}$, which needs to decide whether its input $\gamma \in \{0,1\}^N$ is the output of G (on a random seed) or a random string. $\hat{\mathcal{B}}_{q,i}$ generates $(\mathsf{td}, \mathsf{crs}, \mathsf{pk}_{1-b})$ (and all variables therein) as specified in Sim_1. Then it runs $\beta \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_{1-b})$, simulating $\mathcal{O}_{\mathsf{share}}$ as described next, and outputs β. If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{share}}(\mathsf{pk}_b, (b_i, x_i)_{i\in[n]}, f)$, then $\hat{\mathcal{B}}_{q,i}$ returns $((c_j)_{j\in[n]}, u_{1-b})$, where u_{1-b} is computed as in $\mathbf{H}_2$ and the ciphertexts $c_j = (\bar{c}_j, \pi_j, \mathsf{OTP}_j)$ are generated as follows.

– If fewer than q queries have been made, or if it is the q-th query and $j \leq i$, then $\bar{c}_j \leftarrow \mathsf{HSS.Enc}(\overline{\mathsf{pk}}, 0^\lambda)$; otherwise $\bar{c}_j \leftarrow \mathsf{HSS.Enc}(\overline{\mathsf{pk}}, s_j)$.

– For all queries, $\pi_j \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{b_j}_{\mathsf{NIZK}'}, (\bar{c}_j, \mathsf{pk}))$ for all j.

– If fewer than q queries have been made, or if it is the q-th query and $j < i$, then $\mathsf{OTP}_j \leftarrow \{0,1\}^N$; if it is the q-th query and $j = i$, then $\mathsf{OTP}_i := \gamma \oplus x_i$; otherwise $\mathsf{OTP}_j := G(s_j) \oplus x_j$.

Clearly, the output of $\mathcal{A}$ is distributed as in $\mathbf{H}'$ if $\gamma = G(s_i)$ for some $s_i \leftarrow \{0,1\}^\lambda$, and as in $\mathbf{H}_3^{q,i}$ if $\gamma \leftarrow \{0,1\}^N$. Therefore $\hat{\mathcal{B}}_{q,i}$ and $\mathcal{A}$ have the same distinguishing advantage.

The argument above can also be used to show that $\mathbf{H}_3^{q-1,n_{\mathsf{max}}} \approx \mathbf{H}_3^{q,0}$ for all $0 < q \leq q_{\mathsf{max}}$. Overall we have that the distinguishing advantage of $\mathcal{A}$ between $\mathbf{H}_3^{0,0}$ and $\mathbf{H}_3^{q_{\mathsf{max}},n_{\mathsf{max}}}$ is bounded by

$$q_{\mathsf{max}} \cdot n_{\mathsf{max}} \cdot \max_{q,i} \left\{ \mathsf{Adv}^{\mathsf{hss}}_{\mathsf{HSS},\mathcal{B}_{q,i}}(\lambda) + \mathsf{Adv}^{\mathsf{prg}}_{G,\hat{\mathcal{B}}_{q,i}}(\lambda) \right\},$$

which concludes the proof of reduced share simulatability. We can use a generally similar approach to show that our construction satisfies verifiability. The proof can be found in Appendix B.

Remark 4.3. *A verifiable HSS with non-interactive share reduction (see Definition A.8), as necessary for the secure* 2*-party computation application, can be obtained by combining the techniques of [ADOS22] with the transformation from Section 5. We give an outline of the construction in the following. Let* $\mathbb{F}$ *be a large field and* $\mathcal{F} = \{f : \mathbb{F} \times \mathbb{F} \to \mathbb{F}\}$ *be a function class. Let further* HSS *have* weakly robust linear reconstruction *(this can be obtained from a standard HSS via NIZKs as above and using the transformation of Section 5) for function family* $\mathcal{F}' = \{f'_b : ((\alpha_0, x_0), (\alpha_1, x_1)) \mapsto (f(x_0, x_1), \alpha_b \cdot f(x_0, x_1)) \mid f \in \mathcal{F}, b \in \{0,1\}\}$*. Then, we can obtain a verifiable HSS with non-interactive share reduction for function family* $\mathcal{F}$ *as follows:*

- Encryption. *Party* P_b *chooses* $\alpha_b \leftarrow \mathbb{F}$ *and encrypts* (α_b, x_b)*.*
- Evaluation. *For* $\beta \in \{0,1\}$*, party* P_b *computes shares* $t^\beta_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b, \mathsf{ct}_b)_{b\in\{0,1\}}, f'_\beta)$*.*
- Share reduction. *For* $\beta \in \{0,1\}$*, party* P_b *parses* $t^\beta_b = (y^\beta_b, z^\beta_b, \hat{t}^\beta_b)$*, where* $(y^\beta_b, z^\beta_b) \in \mathbb{F}^2$ *are the additive output shares, and sends the share* (y^{1-b}_b, z^{1-b}_b) *to* P_{1-b}*.*[16]
- Reconstruction. *Party* P_b*, given* $(\alpha_b, (y^b_0, z^b_0), (y^b_1, z^b_1))$*, outputs* $y := y^b_0 + y^b_1$ *if* $z^b_0 + z^b_1 = \alpha_b \cdot y$ *and* $\perp$ *otherwise.*

5 Towards Robust Linear HSS

The starting point for our construction of a malicious HSS scheme in the previous section was a notion we called robust linear reconstruction (Definition 3.3). In this section we will show how this property can be obtained, namely from standard HSS or from homomorphic encryption.

[16]This step does not require knowledge of the output and can thus be written in the form $\mathsf{Reduce}(b, \mathsf{sk}_b, t_b, \perp)$.

5.1 From Standard HSS

We start by giving an intermediary definition of HSS with *non-adaptive robust linear reconstruction*, which more closely captures the standard non-adaptive notion of correctness (except that the reconstruction is required to have a specific structure). In Appendix D we look at how some HSS schemes in the literature fit into this framework.

Definition 5.1 (HSS with Non-Adaptive Robust Linear Reconstruction). *We say a scheme* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval})$ *satisfies* non-adaptive robust linear reconstruction *relative to a function family* $\mathcal{F}$ *if there exists a negligible function* $\epsilon\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *(called the* correctness error*) such that for any (fixed) inputs* $x_1, \ldots, x_n \in I$ *and function* $f \in \mathcal{F}$*, for* $b \in \{0,1\}$*, for* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(b, \mathsf{crs})$*,* $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$*, and for* $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i)$ *for* $i \in [n]$*:*

- *the first component* $\mathsf{sk}[1]$ *of the vector* $\mathsf{sk} := \mathsf{sk}_0 + \mathsf{sk}_1 \in \mathbb{Z}_q^k$ *is* 1*;*
- $\mathsf{Eval}(0, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_0, (\mathsf{ct}_i)_i, f) + \mathsf{Eval}(1, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_1, (\mathsf{ct}_i)_i, f) = f(x_1, \ldots, x_n) \cdot \mathsf{sk}$*,*

except with probability $\epsilon(\lambda)$ *taken over the random coins of* Setup*,* KeyGen *and* Enc*.*

We say that an HSS satisfies weakly non-adaptive robust linear reconstruction *relative to a function family* $\mathcal{F}$*, if the above holds with*

- $\mathsf{Eval}(0, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_0, (\mathsf{ct}_i)_i, f) + \mathsf{Eval}(1, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_1, (\mathsf{ct}_i)_i, f) = f(x_1, \ldots, x_n)$*.*

We now present our transformation from non-adaptive to adaptive correctness, based on techniques from [DNR04]. Note that we impose a strong requirement on the correctness error of the HSS from which we start. To achieve this, the HSS parameters will typically scale (logarithmically) with the size of the input and function spaces. This is a problem as, by the compactness requirement of HSS, shares should be independent of the size $|f|$ (measured in number of operations) of the function to be evaluated and for typical function classes we have $|f| \approx \log |\mathcal{F}|$. One way to mitigate this drawback is to restrict the function class; at best, one can remove the dependency on $|\mathcal{F}|$ by considering evaluation of a polynomially-large class of functions, indpendent of $|f|$, which is typically not an issue in applications, as in many settings the computation to be performed is fixed or agreed upon before the parties set up the parameters. The below transformation therefore only gives an HSS with compactness for *small* function classes $\mathcal{F}$.

Theorem 5.2. *Let* HSS *be a scheme that satisfies (weakly) non-adaptive linear reconstruction relative to function family* $\mathcal{F}$ *and input space* I *with correctness error* $\epsilon(\lambda) \leq 2^{-4\lambda - \log|\mathcal{F}| - n(\log|I| + \lambda)}$*, and where* KeyGen *and* Enc *take at most* $\ell(\lambda)$ *random coins as input. Let* $G\colon \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ *be a secure pseudorandom generator. Then, there exists a (stateful) HSS scheme* HSS' *with (weakly) robust linear reconstruction relative to* $\mathcal{F}$ *(as per Definition 3.3).*

Proof. Without loss of generality we can assume that the algorithms KeyGen and Enc take exactly $\ell(\lambda)$ random coins (as they can ignore any additional ones). We construct HSS as follows (where we only state changed algorithms):

- $\mathsf{Setup}'(1^\lambda)$: Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$. Generate $R_0, R_1, r_1, \ldots, r_n \leftarrow \{0,1\}^{\ell(\lambda)}$. Output $\mathsf{crs}' := (\mathsf{crs}, R_0, R_1, r_1, \ldots, r_n)$.
- $\mathsf{KeyGen}'(b, \mathsf{crs})$: Parse $\mathsf{crs}' =: (\mathsf{crs}, R_0, R_1, r_1, \ldots, r_n)$. Sample $S_b \leftarrow \{0,1\}^\lambda$. Generate $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; G(S_b) \oplus R_b)$.
- $\mathsf{Enc}'(b, \mathsf{crs}, \mathsf{pk}, x_i)$: Parse $\mathsf{crs}' =: (\mathsf{crs}, R_0, R_1, r_1, \ldots, r_n)$. Sample $s_i \leftarrow \{0,1\}^\lambda$. Generate $\mathsf{ct} \leftarrow \mathsf{Enc}(b, \mathsf{crs}, \mathsf{pk}, x_i; G(s_i) \oplus r_i)$.

It is straightforward to see this transformation does not affect security guarantees of the HSS, as an attacker on the security of HSS' could be transformed into either an attacker on the pseudorandomness of G or the security of HSS. It is left to show that HSS' indeed satisfies robust linear reconstruction as required. First, note that, for any fixed function $f \in \mathcal{F}$ and inputs $x_1, \ldots, x_n \in I$, a correctness error in HSS with probability more than $2^\lambda \cdot \epsilon(\lambda)$ can occur for at most a $2^{-\lambda}$ fraction of the setup parameters crs (as otherwise the scheme would have a higher correctness error overall). For the remaining "good" setup parameters crs the error probability is at most $2^\lambda \cdot \epsilon(\lambda)$, where the probability is taken over the randomness of KeyGen and Enc. Now, taking a union bound over all random strings $S_0, S_1, s_1, \ldots, s_n \in \{0,1\}^\lambda$ and all possible inputs $x_1, \ldots, x_n \in I$ and functions $f \in \mathcal{F}$ (this is necessary since which random strings cause an error may depend on the inputs and function being evaluated), we obtain that a correctness error on *any* input for *any* randomness occurs in HSS' at most with probability $2^{\lambda} \cdot \epsilon(\lambda) \cdot 2^{2\lambda + \log|\mathcal{F}| + n\cdot(\log|I|+\lambda)} \leq 2^{-\lambda}$. We thus obtain that HSS' satisfies robust linear reconstruction (with error probability $2^{-\lambda+1}$ over the choice of the setup).

5.2 From Homomorphic Encryption

In this section we will explain how we can use a (leveled) homomorphic encryption scheme to obtain malicious HSS. To that end, we define the *special nearly linear decryption property* of LHE. For instance, the GSW scheme [GSW13] can be adapted to fit into our framework and satisfy this property. We describe how that can be done in Appendix C.

Definition 5.3 (Special nearly linear decryption). *We say* $\mathsf{HE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *has* (d, k, p, q, B)-special nearly linear decryption *if for all* $\lambda \in \mathbb{N}$*, for all public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$*, for all key pairs* $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(b, \mathsf{pp})$*, for all public keys* $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{pp}, \mathsf{pk}_0, \mathsf{pk}_1)$*, for all inputs* $x_1, \ldots, x_\ell \in \{0,1\}$*, for all encryptions* $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i)$ *of* x_i*, for arbitrary polynomial-sized circuits* $C\colon \{0,1\}^\ell \to \{0,1\}$ *of depth at most* d*, and all ciphertexts* $\mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell, C)$ *it holds that:*

- *the first component of the vector* $\mathsf{sk} := \mathsf{sk}_0 + \mathsf{sk}_1 \in \mathbb{Z}_q^k$ *is 1;*
- $\mathsf{ct}' \in \mathbb{Z}_q^{k\times k}$*;*
- $\mathsf{sk}^\top \cdot \mathsf{ct}' = \frac{q}{p} \cdot C(x_1, \ldots, x_\ell) \cdot \mathsf{sk} + e \mod q$ *for some error vector* $e \in \mathbb{Z}_q^k$ *which is bounded by* $\|e\|_\infty \leq B$*.*

Note that if the scheme satisfies (d, k, p, q, B)-special nearly linear decryption for small enough B (namely, $B < \frac{q}{2p}$), then it satisfies perfect correctness for all circuits C of depth at most d, as one can obtain $C(x)$ by rounding the first entry of the vector $\mathsf{ct}' \cdot \mathsf{sk}$ to the closest multiple of q/p.

Since we have at our disposal LHE schemes with perfect correctness, we do not need to use the techniques of the previous section to prevent a malicious adversary from exploiting correctness errors. On the other hand, due to the small noise present when decrypting an LHE ciphertext we cannot achieve robust linear reconstruction exactly as stated in Definition 3.3 – we get only an approximate equality in the last equation of the definition – and for that reason we cannot apply the results from Section 4 directly. The transformation from LHE with special nearly linear decryption into HSS with this "approximate" robust linear reconstruction is actually very straightforward: the HSS algorithms $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}$ are identical to those of LHE, and the evaluation algorithm is defined as follows.

- $\mathsf{Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, C)$: Compute $\overline{\mathsf{ct}}' \leftarrow \mathsf{HE.Eval}((\overline{\mathsf{ct}}_1, \ldots, \overline{\mathsf{ct}}_\ell), C)$ and output $t_b := \overline{\mathsf{ct}}' \cdot \mathsf{sk}_b$.

Using similar techniques as described in Section 4, this can be converted into a malicious HSS scheme. We state our result here and leave the construction to Appendix C.

Theorem 5.4. *Let* $\lambda \in \mathbb{N}$*. Let* N, d', k, p, q, B, B' *(depending on* λ*) be such that* $B \cdot \lambda^{\omega(1)} \leq B' < q/(2p)$*. Let* $G : \{0,1\}^\lambda \to \{0,1\}^N$ *be a* PRG *with depth* d_G *as a boolean circuit. Let* HE *satisfy* (d', k, p, q, B)*-special nearly linear decryption (as defined in Def. 5.3). Let* NIZK *and* NIZK' *be non-interactive zero-knowledge proof of knowledge systems for the families of languages* L_{pp} *and* L'_{pp}*, respectively, as defined in Construction C.1. Then* HSS *as defined in Construction C.1 satisfies reduced share simulatability and verifiability, as per Definitions 3.5 and 3.6, while supporting the evaluation of functions* f *computable by polynomial-sized circuits of depth up to* $d = d' - d_G$*, with one or more inputs in* $\{0,1\}^N$ *and output in* $\{0,1\}$*.*

6 Applications

In this section, we outline applications of malicious HSS to DV-NIZK and malicious secure computation.

6.1 Designated-Verifier NIZK

In this section we show how the HSS properties can be leveraged to create a designated-verifier NIZK scheme. Given that our construction of malicious HSS in this work employs itself a proof system (specifically, a NIZK proof of knowledge), it may seem odd to use such an HSS to construct a DV-NIZK. The main benefit of this approach is that we are able to obtain a *compact* DV-NIZK, even if the underlying NIZK proof of knowledge is not compact. We note also that for this application the aforementioned NIZK proof of knowledge in the preprocessing (CRS) model can be replaced by one in the designated-verifier model in a straightforward manner, though this requires weakening our definition of malicious zero-knowledge (which allows the verifier to generate a public key) to one where the public key is honestly generated during setup (as part of the CRS).[17] We thus obtain a transformation from DV-NIZK to compact DV-NIZK. We present below our DV-NIZK scheme, built directly from malicious HSS.

Construction 6.1. *Fix a security parameter λ and consider the following.*

- *Let $\mathcal{R} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be an NP relation.*
- *Given $\mathcal{R}$, define the function f_x as $f_x(w) := \mathcal{R}(x,w)$ for $x \in \{0,1\}^n$.*
- *Let $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Reduce}, \mathsf{RedComb})$ be a homomorphic secret sharing scheme for the function class $\mathcal{F}_\mathcal{R} := \{f_x : x \in \{0,1\}^n\}$.*

The dvNIZK is constructed as follows.

- $\mathsf{crs} \leftarrow \mathsf{dvNIZK.Setup}(1^\lambda)$*: Generate* $\mathsf{crs}_\mathsf{HSS} \leftarrow \mathsf{HSS.Setup}(1^\lambda)$ *and output* $\mathsf{crs} := \mathsf{crs}_\mathsf{HSS}$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{dvNIZK.KeyGen}(\mathsf{crs})$*: Output* $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{HSS.KeyGen}(1, \mathsf{crs}_\mathsf{HSS})$.
- $\pi \leftarrow \mathsf{dvNIZK.Prove}(\mathsf{crs}, \mathsf{pk}, x, w)$*: Let* $\mathsf{pk}_1 := \mathsf{pk}$. *Sample* $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{HSS.KeyGen}(0, \mathsf{crs}_\mathsf{HSS})$ *and let* $\mathsf{pk}_\mathsf{HSS} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{crs}_\mathsf{HSS}, \mathsf{pk}_0, \mathsf{pk}_1)$. *Next, encrypt the witness as* $\mathsf{ct} \leftarrow \mathsf{HSS.Enc}(0, \mathsf{crs}_\mathsf{HSS}, \mathsf{pk}_\mathsf{HSS}, w)$. *Evaluate* $t_0 \leftarrow \mathsf{HSS.Eval}(0, \mathsf{crs}_\mathsf{HSS}, \mathsf{pk}_\mathsf{HSS}, \mathsf{sk}_0, (0, \mathsf{ct}), f_x)$ *and let* $u_0 \leftarrow \mathsf{HSS.Reduce}(0, \mathsf{sk}_0, t_0, 1)$. *Output the proof*
$$\pi := (\mathsf{pk}_0, \mathsf{ct}, u_0).$$
- $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi)$*: Parse* $\mathsf{pk} =: \mathsf{pk}_1$, $\mathsf{sk} =: \mathsf{sk}_1$, $\pi =: (\mathsf{pk}_0, \mathsf{ct}, u_0)$ *and set the public key* $\mathsf{pk}_\mathsf{HSS} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{crs}_\mathsf{HSS}, \mathsf{pk}_0, \mathsf{pk}_1)$. *Compute* $t_1 \leftarrow \mathsf{HSS.Eval}(1, \mathsf{crs}_\mathsf{HSS}, \mathsf{pk}_\mathsf{HSS}, \mathsf{sk}_1, (0, \mathsf{ct}), f_x)$ *and* $u_1 \leftarrow \mathsf{HSS.Reduce}(1, \mathsf{sk}_1, t_1, 1)$. *Finally, output* 1 *if* $\mathsf{HSS.RedComb}(u_0, u_1) = 1$ *and* 0 *otherwise.*

Proof Length. If HSS has rate 1, then so does dvNIZK. Indeed, for a proof $\pi = (\mathsf{pk}_0, \mathsf{ct}, u_0)$, we have $|\mathsf{pk}_0| = \mathsf{poly}(\lambda)$, $|u_0| = \mathsf{poly}(\lambda)$ and $|\mathsf{ct}| = |w| + \mathsf{poly}(\lambda)$. Hence the total proof length is $|w| + \mathsf{poly}(\lambda)$.

Theorem 6.2. *Let $\mathcal{R}$ be an NP relation and* HSS *a malicious homomorphic secret sharing scheme (as per Definitions 3.2, 3.5 and 3.6) for the function family $\mathcal{F}_\mathcal{R}$. Then* dvNIZK *is a secure designated-verifier non-interactive zero-knowledge proof system for $\mathcal{R}$, as per Definition A.7.*

Completeness. Assume $(\mathsf{pk}_0, \mathsf{ct}, u_0)$ is an honest proof for $(x, w) \in \mathcal{R}$. Then an honest verifier will always accept, since both shares are reduced with respect to the value $1 = f_x(w)$ and thus, by the correctness of HSS, we have $\mathsf{HSS.RedComb}(u_0, u_1) = 1$.

Soundness. In order to prove (restricted) soundness, we consider the following games. Let $x \notin \mathcal{L}$ and suppose an adversary $\mathcal{A}$ breaks the soundness of dvNIZK.

- $\mathbf{H}_0$: This is the real game of the soundness definition, which can be described as follows.
  1. Sample $\mathsf{crs}_\mathsf{HSS} \leftarrow \mathsf{HSS.Setup}(1^\lambda)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HSS.KeyGen}(1, \mathsf{crs}_\mathsf{HSS})$, let $\mathsf{crs} := \mathsf{crs}_\mathsf{HSS}$, $\mathsf{sk} := \mathsf{sk}_1$, and send $(\mathsf{crs}_\mathsf{HSS}, \mathsf{pk}_1)$ to $\mathcal{A}$.
  2. $\mathcal{A}$ makes a polynomial number of verification queries. Given a query (x, π), the challenger answers the query with $\mathsf{dvNIZK.Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi)$. In particular, for a proof $\pi = (\mathsf{pk}_0, \mathsf{ct}, u_0)$, the challenger computes $t_1 \leftarrow \mathsf{HSS.Eval}(1, \mathsf{sk}_1, \mathsf{ct}, f_x)$, $u_1 \leftarrow \mathsf{HSS.Reduce}(1, \mathsf{sk}_1, t_1, 1)$ and outputs 1 if $\mathsf{HSS.RedComb}(u_0, u_1) = 1$ and 0 otherwise.
  3. After the queries, $\mathcal{A}$ outputs π^* and $\mathcal{A}$ wins if $\mathsf{Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi^*) = 1$.

[17] In this setting, only party P_0 (the prover) needs to provide proofs of proper key and ciphertext generation; the other party can be a designated verifier for these proofs.

- $\mathbf{H}_1$: In this game we use some of the simulators from Definition 3.6.

  1. First, generate $(\mathsf{td}_{\mathsf{HSS}}, \mathsf{crs}_{\mathsf{HSS}}) \leftarrow \mathsf{Sim}_{\mathsf{HSS},1}(1^\lambda)$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{HSS.KeyGen}(1, \mathsf{crs}_{\mathsf{HSS}})$, $\mathsf{crs} := \mathsf{crs}_{\mathsf{HSS}}$, $\mathsf{sk} := \mathsf{sk}_1$, and send $(\mathsf{crs}_{\mathsf{HSS}}, \mathsf{pk}_1)$ to $\mathcal{A}$.
  2. $\mathcal{A}$ makes a polynomial number of verification queries. Given a query (x, π), the challenger answers the query with $\overline{\mathsf{Verify}}(\mathsf{crs}, \mathsf{sk}, x, \pi)$, where $\overline{\mathsf{Verify}}$ is defined as follows: compute $w \leftarrow \mathsf{Sim}_{\mathsf{HSS},4}(\mathsf{td}_{\mathsf{HSS}}, \mathsf{pk}_0, \mathsf{ct})$ and accept if and only if $f_x(w) = 1$.
  3. After the queries, $\mathcal{A}$ outputs π^*. We say $\mathcal{A}$ wins if $\overline{\mathsf{Verify}}(\mathsf{crs}, \mathsf{sk}, x, \pi^*) = 1$.

We will use the verifiability property of HSS to show that the difference in the success probability of $\mathcal{A}$ from $\mathbf{H}_0$ to $\mathbf{H}_1$ is negligible. Consider the following HSS adversary $\mathcal{B}$, which attempts to distinguish $\mathsf{Real}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{B},0,f_x}(1^\lambda)$ from $\mathsf{Ideal}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{B},0,f_x}(1^\lambda)$. $\mathcal{B}$ is given input $(\mathsf{crs}_{\mathsf{HSS}}, \mathsf{pk}_1)$ and access to oracles $\mathcal{O}_{\mathsf{ct}}$, $\mathcal{O}_{\mathsf{eval}}$. $\mathcal{B}$ runs $\mathcal{A}$ on input $\mathsf{crs}_{\mathsf{HSS}}$, answering the verification queries of $\mathcal{A}$ as follows: on query (x, π), where $\pi = (\mathsf{pk}_0, \mathsf{ct}, u_0)$, query $u_1 \leftarrow \mathcal{O}_{\mathsf{eval}}(\mathsf{pk}_0, (0, \mathsf{ct}), 1, u_0)$ and output 1 if $u_1 \neq \perp$. When $\mathcal{A}$ outputs π^*, $\mathcal{B}$ similarly queries $u_1 \leftarrow \mathcal{O}_{\mathsf{eval}}(\mathsf{pk}_0, (0, \mathsf{ct}), 1, u_0)$ and outputs 1 if $u_1 \neq \perp$ and 0 otherwise.

Observe that in the real-world HSS experiment $\mathcal{B}$ simulates the view of $\mathcal{A}$ in $\mathbf{H}_0$, while in the ideal-world experiment it simulates the view of $\mathcal{A}$ in $\mathbf{H}_1$. Moreover, in both experiments $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins its respective game. It follows that the difference in success probability between $\mathbf{H}_0$ and $\mathbf{H}_1$ is negligible.

Now, to complete the proof, we show that $\mathcal{A}$ has success probability 0 in $\mathbf{H}_1$. Suppose, by contradiction, that $\mathcal{A}$ outputs π^* such that $\overline{\mathsf{Verify}}(\mathsf{crs}, \mathsf{sk}, x, \pi^*) = 1$. Let $\pi^* = (\mathsf{pk}_0, \mathsf{ct}, u_0)$ and $w^* \leftarrow \mathsf{Sim}_{\mathsf{HSS},4}(\mathsf{td}_{\mathsf{HSS}}, \mathsf{pk}_0, \mathsf{ct})$. Then $1 = f_x(w^*) = \mathcal{R}(x, w^*)$, contradicting the assumption that $x \notin \mathcal{L}$.

Zero knowledge. From the reduced share simulatability of HSS, there exist simulators $\mathsf{Sim}_{\mathsf{HSS},1}$, $\mathsf{Sim}_{\mathsf{HSS},2}$ as in Definition 3.5. We construct the two dvNIZK simulators as follows.

– $\mathsf{Sim}_{\mathsf{dvNIZK},1}(1^\lambda)$: Run $(\mathsf{td}_{\mathsf{HSS}}, \mathsf{crs}_{\mathsf{HSS}}, \mathsf{pk}_0) \leftarrow \mathsf{Sim}_{\mathsf{HSS},1}(1^\lambda)$ and output

$$\mathsf{td} := (\mathsf{td}_{\mathsf{HSS}}, \mathsf{pk}_0),\ \ \mathsf{crs} := \mathsf{crs}_{\mathsf{HSS}}.$$

– $\mathsf{Sim}_{\mathsf{dvNIZK},2}(\mathsf{td}, \mathsf{crs}, \mathsf{pk}, x)$: Parse $\mathsf{td} = (\mathsf{td}_{\mathsf{HSS}}, \mathsf{pk}_0)$ and let $\mathsf{pk}_1 := \mathsf{pk}$. Compute $(\mathsf{ct}, u_0) \leftarrow \mathsf{Sim}_{\mathsf{HSS},2}(\mathsf{td}_{\mathsf{HSS}}, \mathsf{pk}_1, 0, f_x, 1)$. Output $\pi := (\mathsf{pk}_0, \mathsf{ct}, u_0)$.

The result now follows from Definition 3.5, through the following reduction. Let $\mathcal{A}$ be an adversary on the malicious zero-knowledge property of dvNIZK. We define an adversary $\mathcal{B}$ which distinguishes the distributions $\mathsf{Real}^{\mathsf{rss}}_{\mathsf{HSS},\mathcal{A},1,\mathcal{F}}(1^\lambda)$ and $\mathsf{Ideal}^{\mathsf{rss}}_{\mathsf{HSS},\mathcal{A},1,\mathcal{F}}(1^\lambda)$. On input $(\mathsf{crs}, \mathsf{pk}_0)$, $\mathcal{B}$ runs $\mathcal{A}(\mathsf{crs})$, simulating its oracle as follows. On a query (pk, x, w), $\mathcal{B}$ lets $\mathsf{pk}_1 := \mathsf{pk}$ and queries its own oracle on $(\mathsf{pk}_1, (0, w), f_x)$. The oracle outputs (ct, u_0) and $\mathcal{B}$ returns $\pi := (\mathsf{pk}_0, \mathsf{ct}, u_0)$ to $\mathcal{A}$. The output of $\mathcal{B}$ is the output bit of $\mathcal{A}$. Since $\mathcal{B}$ perfectly simulates either the oracle $\mathcal{O}_0$ (in the real experiment) or $\mathcal{O}_1$ (in the ideal experiment), the distinguishing advantage of $\mathcal{A}$ is equal to that of $\mathcal{B}$ and therefore negligible.

The above result gives a rate-1 DV-NIZK from rate-1 malicious HSS. Unfortunately, this is not something we are able to achieve from standard assumptions since, as discussed before, our transformation from HSS with non-adaptive to adaptive linear reconstruction has an overhead which scales with the size of the function and input spaces. What we do achieve by putting together our results from Sections 4, 5, and 6 is the following theorem. Note that here we do not require any compactness properties from the underlying HSS and NIZK and that an HSS scheme with the properties mentioned below can be constructed from the LWE or DCR assumptions if the function class is contained in NC1 (see Appendix D).[18]

Theorem 6.3. *Let $\mathcal{R}$ be an NP relation. If there exist an HSS with non-adaptive robust linear reconstruction for $\mathcal{F}_{\mathcal{R}}$ and NIZK proof-of-knowledge systems for the language families L, L' (cf. Construction 4.1), then there exists a compact DV-NIZK proof system for $\mathcal{R}$.*

[18] We simplify the result by removing the hybrid encryption step (i.e. the use of a PRG) from the malicious HSS construction from Section 4, which is no longer necessary.

6.2 Secure 2-Party Computation

In this section, we construct a maliciously secure two-party computation (2PC) protocol from malicious HSS. Our protocol requires a total of 3 rounds of communication in the standard model (using a CRS).

We consider an adversary who is computationally bounded, malicious (it may instruct a corrupted party to deviate from the protocol) and who may corrupt one of the two parties. We prove that our protocol securely realizes the 2-party function evaluation functionality $\mathcal{F}_{\mathsf{2PC}}$. We formalize it with the assumption that both parties should learn the output of the function evaluation, but the adversary has the ability to selectively abort the protocol, preventing the honest party from receiving the output.[19]

Instantiating with an HSS scheme for $\mathsf{NC1}$ programs with compact shares, the communication complexity of our protocol is $\mathsf{poly}(\lambda) \cdot \log |\mathbb{F}|$, independently of the program size.

Figure 3: Functionality $\mathcal{F}_{\mathsf{2PC}}$

Parameters: A security parameter $\lambda \in \mathbb{N}$, a function $f : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$.

Input phase: The uncorrupted party P_{1-b} provides input $x_{1-b} \in \mathbb{F}$, while the adversary chooses the input $x_b \in \mathbb{F}$ of the corrupted party P_b.

Computation phase: The functionality computes $y := f(x_0, x_1)$.

Selective abort: The adversary receives y and returns either '$\mathsf{continue}$' or 'abort'.

Output phase: In the case of '$\mathsf{continue}$', the uncorrupted party P_{1-b} receives and outputs y; in the case of 'abort', P_{1-b} outputs $\perp$. The corrupted party P_b outputs $\perp$ and the adversary has some arbitrary output.

Figure 4: Protocol for maliciously secure 2-party computation

The common reference string is $\mathsf{crs} \leftarrow \mathsf{HSS.Setup}(1^\lambda)$.

- Party P_b generates $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{HSS.KeyGen}(b, \mathsf{crs})$ and sends pk_b to the other party. Both parties set $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$ as the joint public key.
- Party P_b encrypts their input x_b as $\mathsf{ct}_b \leftarrow \mathsf{HSS.Enc}(b, \mathsf{crs}, \mathsf{pk}, x_b)$ and sends ct_b to the other party.
- Party P_b locally evaluates $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b, \mathsf{ct}_b)_{b \in \{0,1\}}, f)$, computes $u_b \leftarrow \mathsf{HSS.Reduce}(b, \mathsf{sk}_b, t_b, \perp)$ and sends u_b to party P_{1-b}.
- Party P_b on receiving u_{1-b} computes $y \leftarrow \mathsf{RedComb}(u_0, u_1)$ and outputs y.

This protocol has *reusable setup*, in the sense that the first step of the protocol (key generation) is independent of the private inputs x_0, x_1 and the same setup can be reused for multiple different inputs (and multiple functions f, if this is supported by the HSS). We note that HSS with interactive share reduction can also used for the purpose of constructing 2-party computation, at the cost of an extra round in the end (as exchanging shares is only safe if the result indeed verifies, and has therefore to be performed within a secure computation).

Theorem 6.4. *Let $\mathbb{F}$ be a field, $\mathcal{F} = \{f : \mathbb{F} \times \mathbb{F} \to \mathbb{F}\}$ a family of functions, and* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Reduce}, \mathsf{RedComb})$ *a malicious homomorphic secret sharing scheme* with simulatable non-interactive share reduction *(as per Definitions 3.2 and A.8) relative to $\mathcal{F}$. Then, the protocol from Figure 4 securely realizes functionality $\mathcal{F}_{\mathsf{2PC}}$ against a malicious PPT adversary who corrupts at most one party. Further, the setup can be reused for an arbitrary number of evaluations.*

[19] We describe the functionalities $\mathcal{F}_{\mathsf{2PC}}$ in the case of one corruption. If no party is corrupted, then $\mathcal{F}_{\mathsf{2PC}}$ outputs y to both parties.

Proof. Let $\mathcal{A}$ be a (stateful, rushing) real-world adversary which corrupts party P_b for some $b \in \{0,1\}$. Let $(\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$ be the simulator from Definition A.8 (HSS with simulatable non-interactive share reduction). We define an ideal-world adversary $\mathcal{S}$ as follows.

1. $\mathcal{S}$ runs $(\mathsf{td}, \mathsf{crs}, \mathsf{pk}_{1-b}) \leftarrow \mathsf{Sim}_1(1^\lambda)$, and sends crs, pk_{1-b} to $\mathcal{A}$, who returns pk_b.
2. $\mathcal{S}$ computes $\mathsf{ct}_{1-b} \leftarrow \mathsf{Sim}_2(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$ and sends ct_{1-b} to $\mathcal{A}$, who returns ct_b.
3. $\mathcal{S}$ extracts $x_b \leftarrow \mathsf{Sim}_3(\mathsf{td}, \mathsf{pk}_b, \mathsf{ct}_b)$ and sends x_b to $\mathcal{F}_{\mathsf{2PC}}$, which returns $y := f(x_0, x_1)$.
4. $\mathcal{S}$ computes $u_{1-b} \leftarrow \mathsf{Sim}_4(\mathsf{td}, \mathsf{pk}_b, \{(b, \mathsf{ct}_b)\}_{b\in\{0,1\}}, f, y)$. and sends u_{1-b} to $\mathcal{A}$.
5. $\mathcal{S}$ receives u_b from $\mathcal{A}$ and computes $y' \leftarrow \mathsf{RedComb}(u_0, u_1)$. If $y \neq y'$, $\mathcal{S}$ sends 'abort' to $\mathcal{F}_{\mathsf{2PC}}$, otherwise '$\mathsf{continue}$'.

It is straightforward to see that a distinguisher between the real and simulated view of the adversary in this protocol can be transformed into an adversary on the verifiability of HSS with non-interactive share reduction.

Similarly, reusability is a straightforward consequence of Definition A.8 (which allows the adversary to make an arbitrary number of encryption, simulated share and verification queries). This concludes the proof.

Acknowledgements

Pedro Capitão has been supported by the NWO Gravitation Project QSC. Most of this research was carried out while Hila Dahari-Garbian was enrolled as a student at the Weizmann Institute of Science. Hila Dahari-Garbian is supported in part by grants from the Israel Science Foundation (no. 1834/23, and 2337/22), and supported in part by AFOSR Award FA9550-21-1-0046. The work of Lisa Kohl is funded by NWO Talent Programme Veni (VI.Veni.222.348) and by NWO Gravitation project QSC. Part of this work was conducted while the author was visiting the Simons Institute for the Theory of Computing.

References

[ACK23] Thomas Attema, Pedro Capitão, and Lisa Kohl. On homomorphic secret sharing from polynomial-modulus LWE. In *PKC 2023, Part II*, LNCS, pages 3–32, May 2023. 3, 7, 8

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452, August 2022. 3, 4, 5, 6, 7, 8, 9, 16, 36, 37

[BBC+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776. IEEE Computer Society Press, May 2021. 9

[BBC+23] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Arithmetic sketching. In *CRYPTO 2023, Part I*, LNCS, pages 171–202, August 2023. 9

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017. 3

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019. 9

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518, August 2019. 3

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416, August 2020. 9

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018. 3

[BCM23] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. Sublinear-communication secure multiparty computation does not require FHE. In *EUROCRYPT 2023, Part II*, LNCS, pages 159–189, June 2023. 3

[BCM+24] Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Fast public-key silent OT and more from constrained Naor-Reingold. LNCS, pages 88–118, June 2024. 10

[BDS24] Pedro Branco, Nico Döttling, and Akshayaram Srinivasan. Rate-1 statistical non-interactive zero-knowledge. *Cryptology ePrint Archive*, 2024. 10

[BEPU+22] Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. Efficient protocols for oblivious linear function evaluation from ring-lwe. *Journal of Computer Security*, 30(1):39–78, 2022. 3, 8, 9

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367, April 2015. 3

[BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539, August 2016. 3, 9

[BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016. 3, 9

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193, April / May 2017. 3

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33, May 2019. 3, 4, 5, 7, 8, 11, 36

[CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015. 3

[CC18] Pyrros Chaidos and Geoffroy Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 193–221, April / May 2018. 10

[CDH+25] Geoffroy Couteau, Lalita Devadas, Aditya Hegde, Abhishek Jain, and Sacha Servan-Schreiber. Multi-key homomorphic secret sharing. *IACR Cryptol. ePrint Arch.*, page 094, 2025. 9

[CG15] Pyrros Chaidos and Jens Groth. Making sigma-protocols non-interactive without random oracles. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 650–670, March / April 2015. 10

[CGHJ24] Arka Rai Choudhuri, Aarushi Goel, Aditya Hegde, and Abhishek Jain. Homomorphic secret sharing with verifiable evaluation. LNCS, pages 614–650, November 2024. 3, 9

[CH19] Geoffroy Couteau and Dennis Hofheinz. Designated-verifier pseudorandom generators, and their applications. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 562–592, May 2019. 10

[Che23] Xin Chen. Verifiable homomorphic secret sharing for machine learning classifiers. *IEEE Access*, 11:43639–43647, 2023. 3, 5, 9

[CM21] Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 842–870, October 2021. 3, 9

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In *EUROCRYPT 2023, Part III*, LNCS, pages 194–224, June 2023. 3

[COS+22] Ilaria Chillotti, Emmanuela Orsini, Peter Scholl, Nigel Paul Smart, and Barry Van Leeuwen. Scooby: Improved multi-party homomorphic secret sharing based on fhe. In *International Conference on Security and Cryptography for Networks*, pages 540–563. Springer, 2022. 3

[Cou19] Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 473–503, May 2019. 3

[CZ20] Xin Chen and Liang Feng Zhang. Two-server verifiable homomorphic secret sharing for high-degree polynomials. In *Information Security: 23rd International Conference, ISC 2020, Bali, Indonesia, December 16–18, 2020, Proceedings 23*, pages 75–91. Springer, 2020. 3, 9

[dCP22] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 150–179, May / June 2022. 9

[DFN06] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 41–59, March 2006. 10

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122, August 2016. 3, 5, 9

[DIJL23] Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In *CRYPTO 2023, Part II*, LNCS, pages 315–348, August 2023. 3

[DNR04] Cynthia Dwork, Moni Naor, and Omer Reingold. Immunizing encryption schemes from decryption errors. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 342–360, May 2004. 3, 8, 17

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith, III. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017*, volume 10592 of *LNCS*, pages 381–399, October 2017. 3

[GGI+15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *Journal of Cryptology*, 28(4):820–843, October 2015. 4, 7, 10

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658, May 2014. 3

[GRS22] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 323–352, August 2022. 9

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432, April 2008. 28

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92, August 2013. 9, 18, 35

[HZ20] Yan He and Liang Feng Zhang. Cheater-identifiable homomorphic secret sharing for outsourcing computations. *Journal of Ambient Intelligence and Humanized Computing*, 11:5103–5113, 2020. 3, 9

[KNYY19a] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Designated verifier/prover and preprocessing NIZKs from Diffie-Hellman assumptions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 622–651, May 2019. 10

[KNYY19b] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Exploring constructions of compact NIZKs from various assumptions. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 639–669, August 2019. 10

[KNYY20] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Compact NIZKs from standard assumptions on bilinear maps. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 379–409, May 2020. 10

[Lip17] Helger Lipmaa. Optimally sound sigma protocols under DCRA. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 182–203, April 2017. 10

[LPWW20] Benoît Libert, Alain Passelègue, Hoeteck Wee, and David J. Wu. New constructions of statistical NIZKs: Dual-mode DV-NIZKs and more. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 410–441, May 2020. 10

[LTV13] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 94, 2013. 27

[MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. Cryptology ePrint Archive, Report 2025/245, 2025. 4

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708, October 2021. 3, 5, 7, 8, 11, 36

[PsV06] Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. Construction of a non-malleable encryption scheme from any semantically secure one. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 271–289, August 2006. 10

[QRW19] Willy Quach, Ron D. Rothblum, and Daniel Wichs. Reusable designated-verifier NIZKs for all NP from CDH. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 593–621, May 2019. 10, 29

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. 3, 5, 7, 8, 36

[TLM18] Georgia Tsaloli, Bei Liang, and Aikaterini Mitrokotsa. Verifiable homomorphic secret sharing. In *International Conference on Provable Security*, pages 40–55. Springer, 2018. 3, 9

[vBP24] Aron van Baarsen and Sihang Pu. Fuzzy private set intersection with large hyperballs. LNCS, pages 340–369, June 2024. 9

[VV09] Carmine Ventre and Ivan Visconti. Co-sound zero-knowledge with public keys. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 287–304, June 2009. 10

[Wag22] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies*, 2022. 9

[WYG+17] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313, 2017. 3

[YO18] Maki Yoshida and Satoshi Obana. Verifiably multiplicative secret sharing. *IEEE Transactions on Information Theory*, 65(5):3233–3245, 2018. 3, 9

A Additional Definitions

For the sake of completeness and the reader's convenience, we include here some definitions which are relevant in this work.

A.1 Pseudorandom Generator

Definition A.1 (Pseudorandom generator). *Let $\ell \in \mathbb{N}[X]$ be a polynomial. Let G be a polynomial-time algorithm that on input of a seed $s \in \{0,1\}^\lambda$ outputs a string $G(s) \in \{0,1\}^{\ell(\lambda)}$. We say G is a* pseudorandom generator (PRG) *with stretch ℓ, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *such that*

$$\left|\Pr[\mathcal{A}(1^\lambda, G(s)) = 1] - \Pr[\mathcal{A}(1^\lambda, u) = 1]\right| \leq \mathsf{negl}(\lambda)$$

for all $\lambda \in \mathbb{N}$, where the probability is taken over the random choice of $s \leftarrow_R \{0,1\}^\lambda$ and $u \leftarrow_R \{0,1\}^{\ell(\lambda)}$.

A.2 Homomorphic Encryption

We will consider a leveled homomorphic encryption scheme (LHE) that supports 2-party key combination, i.e., the possibility to combine two individually generated public keys $\mathsf{pk}_0, \mathsf{pk}_1$ into a new public key pk, with the parties then holding (additive) shares $\mathsf{sk}_0, \mathsf{sk}_1$ of the corresponding decryption key. We note that this is satisfied by multi-key FHE [LTV13], but is in fact a weaker requirement, as for our purposes it is sufficient that the parties are able to generate encryptions *after* seeing the public key. As we restrict to the 2-party setting in this paper, we will simply refer to this property as key combination in the following and will give all definitions relative to LHE with key combination. The standard syntax of LHE is included in Definition A.5 below.

Definition A.2 (Leveled homomorphic encryption). *A leveled homomorphic encryption scheme* FHE *with 2-party key combination is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *with the following syntax.*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$*:* Setup *is a probabilistic algorithm that takes as input a security parameter λ and depth d and outputs the public parameters* pp.
- $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(b, \mathsf{pp})$*:* KeyGen *is a probabilistic algorithm that takes as input a party index $b \in \{0,1\}$, the public parameters* pp *and outputs a (local) public key* pk_b *and secret key* sk_b.
- $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{pp}, \mathsf{pk}_0, \mathsf{pk}_1)$*:* $\mathsf{KeyComb}$ *is a deterministic algorithm that takes as input the public parameters* pp *and two local public keys* $\mathsf{pk}_0, \mathsf{pk}_1$*, and outputs a joint public key* pk.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$*:* Enc *is a probabilistic algorithm that takes as input a public key* pk *and a message $x \in \{0,1\}$, and outputs a ciphertext* ct.
- $\mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell, C)$*:* Eval *is a deterministic algorithm that takes as input a public key* pk*, ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$ *and a boolean circuit $C : \{0,1\}^\ell \to \{0,1\}$, and outputs an evaluated ciphertext* ct'.
- $\mu \leftarrow \mathsf{Dec}(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{ct}')$*:* Dec *is a deterministic algorithm that takes as input two local secret keys* $\mathsf{sk}_0, \mathsf{sk}_1$ *and a ciphertext* ct'*, and outputs a value μ.*

Definition A.3 (Correctness). *An LHE scheme is* correct *if for every $\lambda, N, d \in \mathbb{N}$, every $x_1, \ldots, x_\ell \in \{0,1\}$ and every circuit $C : \{0,1\}^N \to \{0,1\}$ of depth at most d, it holds that*

$$\Pr[\mathsf{Dec}(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{ct}') \neq C(x_1, \ldots, x_\ell)] = \mathsf{negl}(\lambda),$$

where the probability is over the coins of $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$, $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(b, \mathsf{pp})$ *for $b \in \{0,1\}$,* $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{pp}, \mathsf{pk}_0, \mathsf{pk}_1)$, $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x_i)$ *for $i \in [\ell]$ and* $\mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{ct}, C)$*. If the probability equals 0, we say that the scheme satisfies* perfect correctness.

Definition A.4 (Security). *We say an LHE scheme with key combination* $\mathsf{HE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* secure*, if for any (stateful) PPT adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{>0}$ *such that for any $\lambda, d, N \in \mathbb{N}$, any $b \in \{0,1\}$, for* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$*, for* $(\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}) \leftarrow$

$\mathsf{KeyGen}(\mathsf{pp})$, *for* $(\mathsf{pk}_b, x_0, x_1) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}_{1-b})$ *and for* $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{pp}, \mathsf{pk}_0, \mathsf{pk}_1)$ *with* $\mathsf{pk} \neq \perp$ *it holds that*

$$|\Pr[\mathcal{A}(\mathsf{Enc}(\mathsf{pk}, x_0)) = 1] - \Pr[\mathcal{A}(\mathsf{Enc}(\mathsf{pk}, x_1)) = 1]| \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of Setup*,* KeyGen*,* Enc *and* $\mathcal{A}$*.*

Definition A.5 (Leveled homomorphic encryption - Standard Syntax)**.** *A leveled homomorphic encryption scheme is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *with the following syntax.*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$*:* Setup *is a probabilistic algorithm that takes as input a security parameter* λ *and depth* d *and outputs the public parameters* pp*.*
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*:* KeyGen *is a probabilistic algorithm that takes as input the public parameters* pp *and outputs a public key* pk *and secret key* sk*.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$*:* Enc *is a probabilistic algorithm that takes as input a public key* pk *and a message* $x \in \{0, 1\}$*, and outputs a ciphertext* ct*.*
- $\mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell, C)$*:* Eval *is a deterministic algorithm that takes as input a public key* pk*, ciphertexts* $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$ *and a boolean circuit* $C : \{0, 1\}^\ell \to \{0, 1\}$*, and outputs an evaluated ciphertext* ct'*.*
- $\mu \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}')$*:* Dec *is a deterministic algorithm that takes as input a secret key* sk *and a ciphertext* ct'*, and outputs a value* μ*.*

A.3 Secure Multiparty Computation

In secure multiparty computation (MPC) a set of n parties wish to distributively and securely compute a joint functionality of their inputs $f : \mathcal{X}^n \to \mathcal{Y}$. That is, the parties wish to compute $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$ where party P_i inputs x_i receives back y_i. This task can be easily and trivially solved using a trusted party to whom all parties send their inputs to the trusted party and receive back their outputs. This solution guarantees the *correctness* of the computation along with its security according to several properties such as *privacy* - meaning that the parties learn only their output and what implicitly could be learned from it but nothing else, *independence of inputs* - meaning that the input that is chosen by each party is independent of the other parties' inputs and *guaranteed output delivery* - meaning that the honest parties always receive their outputs. However, in the real world we do not have access to a trusted party and we still wish to achieve the same security properties. Roughly speaking, a secure protocol π realizes functionality f if it allows the parties to securely compute f, even when some of the parties are corrupted. For instance, a corrupted party might try to learn more than its output implicitly reveals, or try to affect the output of a honest party.

The security of a protocol is formally defined by comparing the distribution of the outputs of all parties in a real protocol execution to their outputs when participating in an ideal execution, where a trusted party is given the inputs from the parties, computes f, and returns the outputs. In this ideal execution, a probabilistic polynomial-time (PPT) simulator $\mathcal{S}$ simulates the adversary's view in the protocol's real execution and functions as the adversary in the ideal model (where it only sees the inputs and outputs of the corrupted parties). If the real and ideal output distributions are close, then it follows that the adversary cannot do anything in the real execution that is impossible in the ideal model. Namely, since the ideal model is secure by definition, the protocol is secure as well.

In order to define security, we must define the capability of the adversary, i.e. what the adversary is allowed to do in its attempt to attack the protocol. The main types of adversaries are *semi-honest* (also called honest but curious), who follow the protocol specification but try to learn more than allowed by inspecting the transcript, and *malicious*, who may deviate arbitrarily from the specified protocol in order to break the security (e.g. send modified or new messages that have not been specified by the protocol).

A.4 Non-interactive Zero Knowledge Proofs of Knowledge

In the following we give a definition of non-interactive zero knowledge proofs of knowledge based on [GS08], where we additionally require an extractor that is able to extract the witness.

Definition A.6. *Let* L_{pp} *be a family of NP-languages (for some public parameters* pp*). A non-interactive zero-knowledge proof of knowledge consists of a tuple of PPT algorithms* $\mathsf{NIZK} := (\mathsf{Setup}, \mathsf{TSetup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{Sim}, \mathsf{Ext})$ *of the following syntax.*

- $\mathsf{crs} \leftarrow \mathsf{Setup}(\mathsf{pp})$: *On input the public parameters* pp*, this algorithm generates a common reference string* crs*.*

- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TSetup}(\mathsf{pp})$: *On input the public parameters* pp*, this algorithm generates a common reference string* crs *and additionally a trapdoor* td*.*

- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)$: *On input a common reference string* crs*, statement* x *and witness* w *for* $x \in L_{\mathsf{pp}}$*, this algorithm outputs a proof* π*.*

- $b = \mathsf{Ver}(\mathsf{crs}, x, \pi)$: *On input of a common reference string* crs*, statement* x *and proof* π*, this algorithm outputs a bit* $b \in \{0, 1\}$*.*

- $\pi \leftarrow \mathsf{Sim}(\mathsf{crs}, x)$: *On input a common reference string* crs *and a statement* x*, this algorithm outputs a proof* π*.*

- $w \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{td}, x, \pi)$: *On input a common reference string* crs *with trapdoor* td *and a statement* x *with proof* π*, this algorithm outputs a witness* w*.*

We further require the following to hold.

- ***Completeness:*** *For all public parameters* pp*, for all* $(x, w) \in \mathcal{R}$*,* $\mathsf{crs} \leftarrow \mathsf{Setup}(\mathsf{pp})$*,* $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)$*, we have* $\mathsf{Ver}(\mathsf{crs}, x, \pi) = 1$*.*

- ***Mode indistinguishability:*** *For all public parameters* pp*, we have that the distribution of* $\mathsf{crs} \leftarrow \mathsf{Setup}(\mathsf{pp})$ *and* crs *obtained via* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TSetup}(\mathsf{pp})$ *are computationally indistinguishable.*

- ***Zero knowledge:*** *For all public parameters* pp *and all statements* $x \in L_{\mathsf{pp}}$ *with witness* w *we have that* $\mathsf{Prove}(\mathsf{crs}, x, w)$ *and* $\mathsf{Sim}(\mathsf{crs}, x)$ *are computationally indistinguishable for* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*.*

- ***Knowledge soundness:*** *For all public parameters* pp*, for all PPT adversaries* $\mathcal{A}$*, for all statements* x*, for* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TSetup}(\mathsf{pp})$*, for* $\pi \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{crs}, x)$ *with* $\mathsf{Ver}(\mathsf{crs}, x, \pi) = 1$*, an PPT extractor yields a witness* $w \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{td}, x, \pi)$ *such that* w *is a valid witness for* $x \in L_{\mathsf{pp}}$ *except with negligible probability, where the probability is taken over the random coins of* TSetup*,* $\mathcal{A}$ *and* Ext*.*

A.5 Designated-verifier NIZK

A non-interactive zero-knowledge proof system (NIZK) allows securely proving NP statements in a single message, without the need for several rounds of communication. Formally, a non-interactive protocol between a *prover* and a *verifier* is zero-knowledge if there exists a simulator Sim that can simulate the behavior of every possible corrupted verifier without accessing the prover, such that Sim's output is indistinguishable from the output of the verifier after having interacted with the honest prover. Designated-verifier NIZK (DV-NIZK) is a relaxed model of NIZK, in which prover and verifier have access to a common reference string and the verifier additionally holds a secret key, used to verify proofs. We consider the generation of the CRS and of a public-secret key pair as separate algorithms (as opposed to the traditional definition, where the CRS and secret key are jointly generated by a trusted party and the public key is included in the CRS). This allows us to define the notion of malicious zero knowledge, in which the verifier may maliciously generate the verification key pair, thus obtaining a Malicious Designated-Verifier NIZK [QRW19]. This definition also allows different verifiers to each generate their own public and secret key while reusing the same CRS.

Definition A.7 (DV-NIZK)**.** *Let* $\mathcal{R}$ *be an* NP *relation and* $\mathcal{L}$ *its corresponding language. A DV-NIZK argument for* $\mathcal{L}$ *consists of four efficient algorithms* $\mathsf{dvNIZK} := (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ *with the following syntax.*

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$: *Given a security parameter, the* Setup *algorithm outputs a common reference string* crs*.*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$: *Given* crs*, the* KeyGen *algorithm outputs a public key* pk *and a secret key* sk*.*

- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{pk}, x, w)$: *Given* crs*, the public key* pk*, an instance* x *and a witness* w*, the* Prove *algorithm outputs a proof* π*.*

- $b \leftarrow \mathsf{Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi)$: *Given* crs*, an instance* x*, a proof* π *and the secret verification key* sk*, the* Verify *algorithm outputs a bit* $b \in \{0, 1\}$.

The DV-NIZK should satisfy the following properties.

- *Completeness: For all* $(x, w) \in \mathcal{R}$*, and* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$*, it holds that*

$$\Pr[\mathsf{Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi) = 1 \mid \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{pk}, x, w)] = 1.$$

- *Soundness: For all PPT adversaries* $\mathcal{A}$ *and* $x \notin \mathcal{L}$*,*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathsf{sk}, x, \pi) = 1 \,\middle|\, \pi \leftarrow \mathcal{A}^{\mathsf{Verify}(\mathsf{crs},\mathsf{sk},\cdot,\cdot)}(\mathsf{crs}, \mathsf{pk}, x)\right] = \mathsf{negl}(\lambda),$$

  where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$.

- *Malicious zero knowledge: There exists a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that, for all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr\left[\mathcal{A}^{\mathcal{O}_0(\mathsf{crs},\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_1(\mathsf{td},\widehat{\mathsf{crs}},\cdot,\cdot,\cdot)}(\widehat{\mathsf{crs}}) = 1\right]\right| = \mathsf{negl}(\lambda),$$

  where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(\mathsf{td}, \widehat{\mathsf{crs}}) \leftarrow \mathsf{Sim}_1(1^\lambda)$*. If* $\mathcal{R}(x, w) = 1$, $\mathcal{O}_0(\mathsf{crs}, \mathsf{pk}, x, w)$ *outputs* $\mathsf{Prove}(\mathsf{crs}, \mathsf{pk}, x, w)$*, while* $\mathcal{O}_1(\mathsf{td}, \widehat{\mathsf{crs}}, \mathsf{pk}, x, w)$ *outputs* $\mathsf{Sim}_2(\mathsf{td}, \widehat{\mathsf{crs}}, \mathsf{pk}, x)$*; both oracles output* $\perp$ *if* $\mathcal{R}(x, w) = 0$.

We say that a DV-NIZK argument for a relation $\mathcal{R}$ is *compact* if its proof size is independent of the size (number of gates) of a circuit which computes $\mathcal{R}$.

A.6 HSS Verifiability with Non-interactive Share Reduction

We include here the variant of the definition of verifiability for malicious HSS which we use in our secure 2-party computation protocol.

Definition A.8 (Verifiability with non-interactive share reduction)**.** *We say an HSS scheme* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyComb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Reduce}, \mathsf{RedComb})$ *satisfies* verifiability with non-interactive share reduction *relative to function class* $\mathcal{F}$*, if for all* $b \in \{0, 1\}$ *there exists a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$ *such that for all PPT adversaries* $\mathcal{A}$

$$\left|\Pr\left[\mathsf{Real}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda) = 1\right]\right| \le \mathsf{negl}(\lambda),$$

where the experiments are as defined in Figure 5 and where the probability is taken over the random coins of Setup*,* KeyGen*,* Enc*,* Reduce*,* Sim *and* $\mathcal{A}$.

We sketch here the proof that the HSS scheme outlined in Remark 4.3 satisfies this definition. First, observe that correctness follows from the underlying HSS: if the parties are honest, in the reconstruction step we will have

$$z_0^b + z_1^b = \alpha_b \cdot f(x_0, x_1) = \alpha_b \cdot (y_0^b + y_1^b).$$

Regarding verifiability with non-interactive share reduction, the simulators Sim_1, Sim_2, and Sim_3 can be defined analogously to the ones in the security proof for Construction 4.1 (see Appendix B below). We define the final simulator as follows: $\mathsf{Sim}_4(\mathsf{td}, \mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f, \hat{y})$ extracts (using the NIZK proofs-of-knowledge) the corrupt party's α_b from the respective ciphertext, as well as its key sk_b and evaluates $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$, which is parsed as $t_b = (y_b, z_b, \hat{t}_b)$; then Sim_4 sets $y_{1-b} := \hat{y} - y_b$, $z_{1-b} := \alpha_b \cdot \hat{y} - z_b$ and outputs (y_{1-b}, z_{1-b}). By the knowledge soundness of NIZK and correctness of HSS, the distribution of this reduced share is indistinguishable from one generated honestly (as in $\mathcal{O}^0_{\mathsf{sim}}$).

B Additional Proofs

We complete the proof of Theorem 4.2, by showing that HSS satisfies our notion of verifiability.

$\underline{\mathsf{Real}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda)\text{:}}$
$Q := \emptyset$
$Q_{\mathsf{sim}} := \emptyset$
$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$
$\beta \leftarrow \mathcal{A}^{\mathcal{O}^0_{\mathsf{ct}},\mathcal{O}^0_{\mathsf{sim}},\mathcal{O}^0_{\mathsf{ver}}}(\mathsf{crs}, \mathsf{pk}_{1-b})$
return β

$\underline{\mathcal{O}^0_{\mathsf{ct}}(\mathsf{pk}_b, x)\text{:}}$
$\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$
$\mathsf{ct} \leftarrow \mathsf{Enc}(1-b, \mathsf{crs}, \mathsf{pk}, x)$
$Q := Q \cup \{(\mathsf{pk}_b, \mathsf{ct})\}$
return ct

$\underline{\mathcal{O}^0_{\mathsf{sim}}(\mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)\text{:}}$
$H := \{i \in [n] \mid b_i = 1-b\}$
if $\exists i \in H\colon (\mathsf{pk}_b, \mathsf{ct}_i) \notin Q$ **output** $\bot$
else:
  $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$
  $t_{1-b} \leftarrow \mathsf{Eval}(1-b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_{1-b}, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$
  $u_{1-b} \leftarrow \mathsf{Reduce}(1-b, \mathsf{sk}_{1-b}, t_{1-b}, \bot)$
  $Q_{\mathsf{sim}} := Q_{\mathsf{sim}} \cup \{(\hat{u}_{1-b}, \cdot)\}$
  output u_{1-b}

$\underline{\mathcal{O}^0_{\mathsf{ver}}(u_0, u_1)\text{:}}$
if $(u_{1-b}, \cdot) \notin Q_{\mathsf{sim}}$ **output** $\bot$
else output $\mathsf{RedComb}(u_0, u_1)$

$\underline{\mathsf{Ideal}^{\mathsf{ver}}_{\mathsf{HSS},\mathcal{A},b,\mathcal{F}}(1^\lambda)\text{:}}$
$Q := \emptyset$
$Q_{\mathsf{sim}} := \emptyset$
$(\mathsf{td}, \widehat{\mathsf{crs}}, \widehat{\mathsf{pk}}_{1-b}) \leftarrow \mathsf{Sim}_1(1^\lambda)$
$\beta \leftarrow \mathcal{A}^{\mathcal{O}^1_{\mathsf{ct}},\mathcal{O}^0_{\mathsf{sim}},\mathcal{O}^1_{\mathsf{ver}}}(\widehat{\mathsf{crs}}, \widehat{\mathsf{pk}}_{1-b})$
return β

$\underline{\mathcal{O}^1_{\mathsf{ct}}(\mathsf{pk}_b, x)\text{:}}$
$\mathsf{ct} \leftarrow \mathsf{Sim}_2(\widehat{\mathsf{crs}}, \widehat{\mathsf{pk}}_{1-b}, \mathsf{pk}_b)$
$Q := Q \cup \{(\mathsf{pk}_b, \mathsf{ct}, x)\}$
return ct

$\underline{\mathcal{O}^1_{\mathsf{sim}}(\mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)\text{:}}$
$H := \{i \in [n] \mid b_i = 1-b\}$
if $\exists i \in H\colon (\mathsf{pk}_b, \mathsf{ct}_i, \cdot) \notin Q$ **output** $\bot$
else:
  for all $i \in H$:
    $x_i := x$ **for** $(\mathsf{pk}_b, \mathsf{ct}_i, x) \in Q$
  for all $i \in [n] \setminus H$:
    $x_i \leftarrow \mathsf{Sim}_3(\mathsf{td}, \mathsf{pk}_b, \mathsf{ct}_i)$
  $\hat{y} := f(x_1, \ldots, x_n)$
  $\hat{u}_{1-b} \leftarrow \mathsf{Sim}_4(\mathsf{td}, \mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f, \hat{y})$
  $Q_{\mathsf{sim}} := Q_{\mathsf{sim}} \cup \{(\hat{u}_{1-b}, \hat{y})\}$
  output $\hat{u}_{1-b}$

$\underline{\mathcal{O}^1_{\mathsf{ver}}(u_0, u_1)\text{:}}$
if $\forall \hat{y}\colon\ (u_{1-b}, \hat{y}) \notin Q_{\mathsf{sim}}$ **output** $\bot$
else if $\exists\ \hat{y}\colon\ (u_{1-b}, \hat{y}) \in Q_{\mathsf{sim}} \wedge$
  $\mathsf{RedComb}(u_0, u_1) \neq \hat{y}$ **output** $\bot$
else output $\hat{y}$

Figure 5: HSS verifiability with non-interactive share reduction.

Theorem (HSS Verifiability)**.** *Let $\lambda \in \mathbb{N}$ denote a security parameter. Let $G : \{0,1\}^\lambda \to \{0,1\}^N$ be a* PRG*, with depth d_G as a boolean circuit. Let* $\overline{\mathsf{HSS}}$ *be a robust linear HSS scheme, supporting the evaluation of functions computable by polynomial-sized programs of depth up to d'. Let* NIZK *and* NIZK′ *be non-interactive zero-knowledge proof of knowledge systems for the families of languages $L_{b,\overline{\mathsf{crs}}}$ and $L'_{\overline{\mathsf{crs}}}$, respectively, as defined in Construction 4.1.*

Then HSS *as defined in Construction 4.1 satisfies verifiability, as per Definition 3.6, while supporting the evaluation of functions f computable by polynomial-sized programs of depth up to $d = d' - d_G$, with one or more inputs in $\{0,1\}^N$ and output in $\{0,1\}$.*

Proof. We define the simulators required by the security definition.

– $\mathsf{Sim}_1(1^\lambda)$:

1. Generate $\overline{\mathsf{crs}} \leftarrow \overline{\mathsf{HSS}}.\mathsf{Setup}(1^\lambda)$
2. $(\mathsf{crs}^\beta_{\mathsf{NIZK}}, \mathsf{td}^\beta_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK}.\mathsf{TSetup}(\overline{\mathsf{crs}})$, $\beta \in \{0,1\}$
3. $(\mathsf{crs}^\beta_{\mathsf{NIZK}'}, \mathsf{td}^\beta_{\mathsf{NIZK}'}) \leftarrow \mathsf{NIZK}'.\mathsf{TSetup}(\overline{\mathsf{crs}})$, $\beta \in \{0,1\}$
4. $\mathsf{crs} := (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$,
   $\mathsf{td} := (\mathsf{crs}, \mathsf{td}^0_{\mathsf{NIZK}}, \mathsf{td}^1_{\mathsf{NIZK}}, \mathsf{td}^0_{\mathsf{NIZK}'}, \mathsf{td}^1_{\mathsf{NIZK}'})$
5. $(\overline{\mathsf{pk}}_{1-b}, \overline{\mathsf{sk}}_{1-b}) \leftarrow \overline{\mathsf{HSS}}.\mathsf{KeyGen}(\overline{\mathsf{crs}})$
6. $\pi_{1-b} \leftarrow \mathsf{NIZK}.\mathsf{Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$
7. $\mathsf{pk}_{1-b} := (\overline{\mathsf{pk}}_{1-b}, \pi_{1-b})$
8. Output $(\mathsf{td}, \mathsf{crs}, \mathsf{pk}_{1-b})$.

– $\mathsf{Sim}_2(\mathsf{crs}, \mathsf{pk}_{1-b}, \mathsf{pk}_b)$:

1. Parse $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$, $\mathsf{td} = (\mathsf{crs}, \mathsf{td}^0_{\mathsf{NIZK}}, \mathsf{td}^1_{\mathsf{NIZK}}, \mathsf{td}^0_{\mathsf{NIZK}'}, \mathsf{td}^1_{\mathsf{NIZK}'})$
2. $\mathsf{pk} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{pp}, \mathsf{pk}_{1-b}, \mathsf{pk}_b)$
3. $\overline{\mathsf{ct}} \leftarrow \overline{\mathsf{HSS}}.\mathsf{Enc}(\overline{\mathsf{pk}}, 0^\lambda)$
4. $\pi \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}, \overline{\mathsf{pk}}))$
5. Output $\mathsf{ct} := (\overline{\mathsf{ct}}, \pi, \mathsf{OTP})$, where $\mathsf{OTP} \leftarrow \{0,1\}^N$.

– $\mathsf{Sim}_3(\mathsf{td}, \mathsf{pk}_b, \mathsf{ct}_i)$:

1. Parse $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$, $\mathsf{td} = (\mathsf{crs}, \mathsf{td}^0_{\mathsf{NIZK}}, \mathsf{td}^1_{\mathsf{NIZK}}, \mathsf{td}^0_{\mathsf{NIZK}'}, \mathsf{td}^1_{\mathsf{NIZK}'})$
2. Parse $\mathsf{ct}_i = (\overline{\mathsf{ct}}_i, \pi_i, \mathsf{OTP}_i)$
3. $(s_i, r_i) \leftarrow \mathsf{NIZK}'.\mathsf{Ext}(\mathsf{crs}^{b_i}_{\mathsf{NIZK}'}, \mathsf{td}^{b_i}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}_i, \mathsf{pk}), \pi_i)$
4. Output $x_i := \mathsf{OTP}_i \oplus G(s_i)$.

– $\mathsf{Sim}_4(\mathsf{td}, \mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f, y, u_b)$:

1. Parse $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$, $\mathsf{td} = (\mathsf{crs}, \mathsf{td}^0_{\mathsf{NIZK}}, \mathsf{td}^1_{\mathsf{NIZK}}, \mathsf{td}^0_{\mathsf{NIZK}'}, \mathsf{td}^1_{\mathsf{NIZK}'})$
2. Parse $(\overline{\mathsf{pk}}_b, \pi_b) \leftarrow \mathsf{pk}_b$
3. $(\mathsf{sk}_b, r) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}^b_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_b, \pi_b)$
4. $\mathsf{pk} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{pp}, \mathsf{pk}_{1-b}, \mathsf{pk}_b)$
5. $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$
6. $u_{1-b} := (y, y \cdot \mathsf{sk}_b - t_b)$
7. If $\mathsf{HSS.RedComb}(u_0, u_1) \neq y$, output $\perp$; otherwise output u_{1-b}.

Now we define a sequence of indistinguishable hybrid games.

- $\mathbf{H}_0$: This is the real game described in Definition 3.6, with output $\mathcal{A}^{\mathcal{O}^0_{\mathsf{ct}}, \mathcal{O}^0_{\mathsf{ver}}}(\mathsf{crs}, \mathsf{pk}_{1-b})$, where the oracles are as specified in the definition, $\mathsf{crs} \leftarrow \mathsf{HSS.Setup}(1^\lambda)$ and $(\mathsf{pk}_{1-b}, \mathsf{sk}_{1-b}) \leftarrow \mathsf{HSS.KeyGen}(\mathsf{crs})$.

- $\mathbf{H}_1$: The same as $\mathbf{H}_0$, but now we generate the crs as in Sim_1, with $(\mathsf{crs}^\beta_{\mathsf{NIZK}}, \mathsf{td}^\beta_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TSetup}(\overline{\mathsf{crs}})$, $(\mathsf{crs}^\beta_{\mathsf{NIZK}'}, \mathsf{td}^\beta_{\mathsf{NIZK}'}) \leftarrow \mathsf{NIZK}'.\mathsf{TSetup}(\overline{\mathsf{crs}})$ for $\beta \in \{0,1\}$.

- $\mathbf{H}_2$: In this hybrid we replace the proof of correct public key generation with a simulated one: $\mathsf{pk}_{1-b} := (\overline{\mathsf{pk}}_{1-b}, \pi_{1-b})$, where $(\overline{\mathsf{pk}}_{1-b}, \overline{\mathsf{sk}}_{1-b}) \leftarrow \mathsf{HSS.KeyGen}(\mathsf{crs})$, $\pi_{1-b} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$. We do the same for the ciphertext in the oracle $\mathcal{O}_{\mathsf{ct}}$. The proof included in the ciphertext $\mathsf{ct} = (\overline{\mathsf{ct}}, \pi, \mathsf{OTP})$ is now $\pi \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}, \overline{\mathsf{pk}}))$. Additionally, we register the inputs x queried to $\mathcal{O}_{\mathsf{ct}}$, i.e. we set $Q := Q \cup \{(\mathsf{pk}_b, \mathsf{ct}, x)\}$ on each query.

- $\mathbf{H}_3$: In this hybrid we change the definition of $\mathcal{O}_{\mathsf{ver}}$. First, we check whether $y = f(x_1, \ldots, x_n)$, where $x_i := x$ such that $(\mathsf{pk}_b, \mathsf{ct}_i, x) \in Q$, if $i \in H$, and $x_i \leftarrow \mathsf{Sim}_3(\mathsf{td}, \mathsf{pk}_b, \mathsf{ct}_i)$, if $i \in [n] \setminus H$ (as in the ideal world execution). If $y \neq f(x_1, \ldots, x_n)$, the execution of $\mathcal{O}_{\mathsf{ver}}$ remains unchanged. If $y = f(x_1, \ldots, x_n)$, we change the way in which u_{1-b} is computed: instead of "$u_{1-b} \leftarrow \mathsf{HSS.Reduce}(1-b, \mathsf{sk}_{1-b}, t_{1-b}, y)$", we do the following. Parse $(\overline{\mathsf{pk}}_b, \pi_b) \leftarrow \mathsf{pk}_b$, extract $(\mathsf{sk}_b, r) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}^b_{\mathsf{NIZK}}, \mathsf{td}^b_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_b, \pi_b)$, evaluate $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$ and let $u_{1-b} := (y, y \cdot \mathsf{sk}_b - t_b)$. If $\mathsf{HSS.RedComb}(u_0, u_1) \neq y$, output $\perp$; otherwise output u_{1-b}. Notice that this is identical to Sim_4.

- $\mathbf{H}_4$: In this hybrid we add the condition "**if** $f(x_1, \ldots, x_n) \neq y$ **output** $\perp$" to $\mathcal{O}_{\mathsf{ver}}$ in the previous hybrid, where again $x_i := x$ such that $(\mathsf{pk}_b, \mathsf{ct}_i, x) \in Q$, if $i \in H$, and $x_i \leftarrow \mathsf{Sim}_3(\mathsf{td}, \mathsf{pk}_b, \mathsf{ct}_i)$, if $i \in [n] \setminus H$.

- $\mathbf{H}_5$: The same as $\mathbf{H}_4$, but in $\mathcal{O}_{\mathsf{ct}}$, instead of encrypting s, we encrypt 0^λ. Specifically, the ciphertext becomes $\mathsf{ct} := (\overline{\mathsf{ct}}, \pi, \mathsf{OTP})$, where $\overline{\mathsf{ct}} \leftarrow \overline{\mathsf{HSS}}.\mathsf{Enc}(\overline{\mathsf{pk}}, 0^\lambda)$, $\pi \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}, \overline{\mathsf{pk}}))$ and $\mathsf{OTP} := G(s) \oplus x_\beta$.

- $\mathbf{H}_6$: Identical to the previous hybrid, but in $\mathcal{O}_{\mathsf{ct}}$, instead of computing $\mathsf{OTP} := G(s) \oplus x_\beta$, we randomly sample $\mathsf{OTP} \leftarrow \{0,1\}^N$.

We show that each hybrid is computationally indistinguishable from the next.

$\mathbf{H}_0 \approx \mathbf{H}_1$: By the properties of NIZK and NIZK', the crs generated by Setup or TSetup are indistinguishable.

$\mathbf{H}_1 \approx \mathbf{H}_2$: The simulated proof $\pi_{1-b} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$ is indistinguishable from $\pi_{1-b} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b}, (\overline{\mathsf{pk}}_{1-b}, r))$, by the zero-knowledge property of NIZK. Similarly, $\pi \leftarrow \mathsf{NIZK}'.\mathsf{Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}, \overline{\mathsf{pk}}))$ and $\pi \leftarrow \mathsf{NIZK}'.\mathsf{Prove}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}, \overline{\mathsf{pk}}), (s, r))$ are computationally indistinguishable. It follows that an adversary cannot distinguish the game with the simulated proofs from the game with the real proofs.

$\mathbf{H}_2 \approx \mathbf{H}_3$: The output of $\mathcal{O}_{\mathsf{ver}}$ is identical in both hybrids in the case $y \neq f(x_1, \dots, x_n)$. We will now show that, in the case $y = f(x_1, \dots, x_n)$, the output of $\mathcal{O}_{\mathsf{ver}}$ is statistically indistinguishable. Let us assume that, for all $i \in [n]$, the ciphertexts $\mathsf{ct}_i = (\overline{\mathsf{ct}}_i, \pi_i, \mathsf{OTP}_i)$ are such that $\mathsf{NIZK}'.\mathsf{Verify}(\mathsf{crs}^{b_i}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}_i, \mathsf{pk}), \pi_i) = 1$ (otherwise the oracle outputs $\perp$ in either hybrid). By knowledge soundness of NIZK', for $i \notin H$ we have that the inputs $x_i \leftarrow \mathsf{Sim}_3(\mathsf{td}, \mathsf{pk}_b, \mathsf{ct}_i)$ are of the form $x_i = \mathsf{OTP}_i \oplus G(s_i)$ with $\overline{\mathsf{ct}}_i = \overline{\mathsf{HSS}}.\mathsf{Enc}(\mathsf{pk}, s_i; r_i)$. The same holds for $i \in H$, since the corresponding encryptions are generated honestly. Similarly, by knowledge soundness of NIZK, the extracted key sk_b is valid: $(\overline{\mathsf{pk}}_b, \mathsf{sk}_b) = \overline{\mathsf{HSS}}.\mathsf{KeyGen}(\overline{\mathsf{crs}}; r)$ for some randomness r.

As in the previous proof, by the robust linear reconstruction property of $\overline{\mathsf{HSS}}$ we have

$$t_b + t_{1-b} = f'(s_1, \dots, s_n) \cdot (\mathsf{sk}_b + \mathsf{sk}_{1-b}).$$

Since $f'(s_1, \dots, s_n) = y$, we can rewrite this as

$$t_{1-b} - y \cdot \mathsf{sk}_{1-b} = y \cdot \mathsf{sk}_b - t_b,$$

which shows that $\tilde{u}_{1-b} := t_{1-b} - y \cdot \mathsf{sk}_{1-b}$ (as defined in $\mathbf{H}_2$) and $\tilde{u}_{1-b} := y \cdot \mathsf{sk}_b - t_b$ (as defined in $\mathbf{H}_3$) are identical.

$\mathbf{H}_3 \approx \mathbf{H}_4$: It suffices to show that, whenever a query to $\mathcal{O}_{\mathsf{ver}}$ is such that $f(x_1, \dots, x_n) \neq y$, the output of $\mathcal{O}_{\mathsf{ver}}$ in $\mathbf{H}_3$ is $\perp$. Assume by contradiction that there exists an adversary $\mathcal{A}$ which on its j-th query to $\mathcal{O}_{\mathsf{ver}}$ obtains, with non-negligible probability, an output $u_{1-b} \neq \perp$ when $f(x_1, \dots, x_n) \neq y$. We construct a PPT adversary $\mathcal{B}$ that breaks the security of $\overline{\mathsf{HSS}}$ with non-negligible probability. Given $(\overline{\mathsf{crs}}, \overline{\mathsf{pk}}_{1-b})$ from the challenger, $\mathcal{B}$ runs $(\mathsf{crs}^{\beta}_{\mathsf{NIZK}}, \mathsf{td}^{\beta}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TSetup}(\overline{\mathsf{crs}})$ and $(\mathsf{crs}^{\beta}_{\mathsf{NIZK}'}, \mathsf{td}^{\beta}_{\mathsf{NIZK}'}) \leftarrow \mathsf{NIZK}'.\mathsf{TSetup}(\overline{\mathsf{crs}})$ for $\beta \in \{0,1\}$. $\mathcal{B}$ generates the simulated proof $\pi_{1-b} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$, sends $(\mathsf{crs}, \mathsf{pk}_{1-b})$ to $\mathcal{A}$, where $\mathsf{pk}_{1-b} := (\overline{\mathsf{pk}}_{1-b}, \pi_{1-b})$, and sets $Q = \emptyset$.

$\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ct}}$ access to $\mathcal{A}$ as follows: given a query (pk_b, x), $\mathcal{B}$ computes $\mathsf{pk} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{crs}, \mathsf{pk}_b, \mathsf{pk}_{1-b})$, sets $Q = Q \cup \{(\mathsf{pk}_b, \mathsf{ct}, x)\}$, and returns $c \leftarrow \mathsf{HSS.Enc}(1-b, \mathsf{crs}, \mathsf{pk}, x)$.

For the first $j-1$ queries, $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ver}}$ access to $\mathcal{A}$ as follows. Given a query $(\mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i \in [n]}, y, u_b)$, $\mathcal{B}$ checks whether $y = f(x_1, \dots, x_n)$, where $x_i := x$ such that $(\mathsf{pk}_b, \mathsf{ct}_i, x) \in Q$, if $i \in H$, and $x_i \leftarrow \mathsf{Sim}_3(\mathsf{td}, \mathsf{pk}_b, \mathsf{ct}_i)$, if $i \in [n] \setminus H$. If $y = f(x_1, \dots, x_n)$, then $\mathcal{B}$ computes the output of the oracle as described in $\mathbf{H}_3$. If $y \neq f(x_1, \dots, x_n)$, $\mathcal{B}$ simply outputs $\perp$.

For the j-th query $(\mathsf{pk}_b, (b_i, \mathsf{ct}_i)_{i \in [n]}, y, u_b)$, $\mathcal{B}$ parses $\mathsf{pk}_b = (\overline{\mathsf{pk}}_b, \pi_b)$, extracts $(\mathsf{sk}_b, r) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}^b_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_b, \pi_b)$, and sets $\mathsf{pk} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{crs}, \mathsf{pk}_b, \mathsf{pk}_{1-b})$. Additionally, $\mathcal{B}$ computes the share $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i \in [n]}, f)$ and defines

$$\hat{\mathsf{sk}}_{1-b} := (2y-1) \cdot \big(\tilde{u}_b - t_b + f(x_1, \dots, x_n) \cdot \mathsf{sk}_b\big),$$

where u_b is parsed as $u_b =: (y, \tilde{u}_b)$. Finally, $\mathcal{B}$ checks the validity of the key pair $(\mathsf{pk}, \mathsf{sk})$ with respect to the $\overline{\mathsf{HSS}}$ parameters $\overline{\mathsf{crs}}$, where $\mathsf{sk} := \hat{\mathsf{sk}}_{1-b} + \mathsf{sk}_b$. If it is a valid key pair, $\mathcal{B}$ sends to the challenger $m_0 = 0, m_1 = 1$, receives back a ciphertext $\overline{c}$, decrypts it using sk and outputs the resulting message. If $(\mathsf{pk}, \mathsf{sk})$ is not a valid $\overline{\mathsf{HSS}}$ key pair, or if $\mathcal{A}$ made fewer than j queries to $\mathcal{O}_{\mathsf{ver}}$, then $\mathcal{B}$ outputs a random bit.

Let us now check that, assuming the j-th query of $\mathcal{A}$ to $\mathcal{O}_{\mathsf{ver}}$ is the first such that $y \neq f(x_1, \ldots, x_n)$ and the output is not $\perp$, $\mathcal{B}$ succeeds in the $\overline{\mathsf{HSS}}$ security game with probability 1. Note that the first $j-1$ queries to $\mathcal{O}_{\mathsf{ver}}$ perfectly simulate the behavior of the oracle in $\mathbf{H}_3$. Since the j-th query of $\mathcal{A}$ would have resulted in a non-$\perp$ output, we have $\mathsf{HSS.RedComb}(u_0, u_1) = y$, which means $\tilde{u}_b + \tilde{u}_{1-b} = 0$. Here

$$\tilde{u}_{1-b} := \mathsf{HSS.Reduce}(1-b, \mathsf{sk}_{1-b}, t_{1-b}, y) = t_{1-b} - (1 - f(x_1, \ldots, x_n)) \cdot \mathsf{sk}_{1-b},$$

since $y = 1 - f(x_1, \ldots, x_n)$. Therefore

$$(1 - f(x_1, \ldots, x_n)) \cdot \mathsf{sk}_{1-b} = t_{1-b} + \tilde{u}_b.$$

Meanwhile, by the robust linear reconstruction property of $\overline{\mathsf{HSS}}$ we have

$$t_b + t_{1-b} = f(x_1, \ldots, x_n) \cdot (\mathsf{sk}_b + \mathsf{sk}_{1-b}).$$

From the last two equations we get

$$(1 - 2f(x_1, \ldots, x_n)) \cdot \mathsf{sk}_{1-b} = \tilde{u}_b - t_b + f(x_1, \ldots, x_n) \cdot \mathsf{sk}_b.$$

Noting that $1 - 2f(x_1, \ldots, x_n) = 2y - 1 \in \{\pm 1\}$, we have that $\hat{\mathsf{sk}}_{1-b} = \mathsf{sk}_{1-b}$, i.e. $\mathcal{B}$ recovers the secret key of party $1-b$ and is able to correctly decrypt. Since this event occurs with non-negligible probability, and the guessing probability of $\mathcal{B}$ is at least $1/2$ when the event does not occur, $\mathcal{B}$ has non-negligible advantage in the $\overline{\mathsf{HSS}}$ security game.

$\mathbf{H}_4 \approx \mathbf{H}_5$: We show a reduction to the security of $\overline{\mathsf{HSS}}$. For simplicity, we assume $\mathcal{A}$ makes a single call to the oracle $\mathcal{O}_{\mathsf{ct}}$; this can be generalized to any polynomial number of queries via a standard argument, as was done in the previous proof. Consider the following PPT adversary $\mathcal{B}$. On input $(\overline{\mathsf{crs}}, \overline{\mathsf{pk}}_{1-b})$, it generates $(\mathsf{crs}^{\beta}_{\mathsf{NIZK}}, \mathsf{td}^{\beta}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TSetup}(\overline{\mathsf{crs}})$, $(\mathsf{crs}^{\beta}_{\mathsf{NIZK'}}, \mathsf{td}^{\beta}_{\mathsf{NIZK'}}) \leftarrow \mathsf{NIZK'.TSetup}(\overline{\mathsf{crs}})$ for $\beta \in \{0,1\}$, and sets $\mathsf{crs} = (\overline{\mathsf{crs}}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK'}}, \mathsf{crs}^1_{\mathsf{NIZK'}})$. Then $\mathcal{B}$ lets $\mathsf{pk}_{1-b} = (\overline{\mathsf{pk}}_{1-b}, \pi_{1-b})$, where $\pi_{1-b} \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_{1-b})$, and runs $\beta \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_{1-b})$, simulating the oracles as follows. Whenever $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{ver}}$, $\mathcal{B}$ computes the response as the in definition of $\mathcal{O}^1_{\mathsf{ver}}$ in Figure 2 (note that this does not require knowledge of the secret key sk_{1-b}). When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{ct}}(\mathsf{pk}_b, x)$, $\mathcal{B}$ randomly chooses $s \leftarrow \{0,1\}^{\lambda}$ and sets $m_0 = s, m_1 = 0^{\lambda}$. $\mathcal{B}$ sends to the challenger $(\mathsf{pk}_b, m_0, m_1)$, and receives back a ciphertext $\overline{\mathsf{ct}}$. Then $\mathcal{B}$ sends $\mathsf{ct} := (\overline{\mathsf{ct}}, \pi, \mathsf{OTP})$ to $\mathcal{A}$, where $\pi \leftarrow \mathsf{NIZK'.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK'}}, (\overline{\mathsf{ct}}, \overline{\mathsf{pk}}))$ and $\mathsf{OTP} := G(s) \oplus x$. Finally, $\mathcal{B}$ outputs β, the output of $\mathcal{A}$.

If $\overline{\mathsf{ct}}$ is an encryption of m_0, then (in the view of $\mathcal{A}$) the adversary $\mathcal{B}$ simulates the hybrid $\mathbf{H}_4$; otherwise $\mathcal{B}$ simulates the hybrid $\mathbf{H}_5$. From the security of $\overline{\mathsf{HSS}}$, we have that the probability that $\mathcal{B}$ can distinguish between the ciphertexts is negligible, which implies that the probability that $\mathcal{A}$ can distinguish between the hybrids is also negligible.

$\mathbf{H}_5 \approx \mathbf{H}_6$: We show a reduction to the pseudorandomness property of G. Consider a PPT adversary $\mathcal{B}$ which on input γ needs to decide whether it is a PRG output or a random string. $\mathcal{B}$ generates $(\mathsf{td}, \mathsf{crs}, \mathsf{pk}_{1-b}) \leftarrow \mathsf{Sim}_1(1^{\lambda})$, runs $\beta \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{pk}_{1-b})$ and outputs β. The oracle $\mathcal{O}_{\mathsf{ver}}$ is simulated as in the previous hybrid. Once more we will assume for simplicity that there is a single query to the oracle $\mathcal{O}_{\mathsf{ct}}$. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{ct}}(\mathsf{pk}_b, x)$, then $\mathcal{B}$ runs $\mathsf{pk} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$, computes $\overline{\mathsf{ct}} \leftarrow \overline{\mathsf{HSS}}.\mathsf{Enc}(\overline{\mathsf{pk}}, 0^{\lambda})$, $\pi \leftarrow \mathsf{NIZK'.Sim}(\mathsf{crs}^{1-b}_{\mathsf{NIZK'}}, (\overline{\mathsf{ct}}, \overline{\mathsf{pk}}))$, $\mathsf{OTP} := \gamma \oplus x$, and sends $(\overline{\mathsf{ct}}, \pi, \mathsf{OTP})$ to $\mathcal{A}$.

Observe that if γ is a PRG output then $\mathcal{B}$ simulates, in the view of $\mathcal{A}$, hybrid $\mathbf{H}_4$; otherwise, $\mathcal{B}$ simulates hybrid $\mathbf{H}_5$. By the security of the PRG, the distinguishing advantage is negligible.

C Malicious HSS from Homomorphic Encryption

We present here the construction of a maliciously secure HSS scheme from a leveled homomorphic encryption scheme with special nearly linear decryption (see Definition 5.3). In essence, it is obtained by first transforming the LHE scheme into an HSS with robust linear reconstruction and then converting that into a malicious HSS, using the techniques described in Sections 5.2 and 4, respectively. The main additional challenge is the presence of small error terms in decryption, which we address by using a *noise flooding* technique.

Construction C.1. *Let $\lambda \in \mathbb{N}$. Let N, d', k, p, q, B, B' (depending on λ) be such that $B \cdot \lambda^{\omega(1)} \leq B' < q/(2p)$. Let $G : \{0,1\}^\lambda \to \{0,1\}^N$ be a* PRG *with depth d_G as a boolean circuit. Let* HE $=$ (Setup, KeyGen, KeyComb, Enc, Eval, Dec) *satisfy (d', k, p, q, B)-special nearly linear decryption (as defined in Def. 5.3). Let* NIZK $=$ (Setup, TSetup, Prove, Ver, Sim, Ext) *be a NIZK proof of knowledge for the family of languages*

$$L_{b,\mathsf{pp}} = \{\overline{\mathsf{pk}}_b \mid \exists(\overline{\mathsf{sk}}_b, r) \text{ such that } (\overline{\mathsf{pk}}_b, \overline{\mathsf{sk}}_b) = \mathsf{HE.KeyGen}(b, \mathsf{pp}; r)\}.$$

Let $\mathsf{NIZK}' = (\mathsf{Setup}', \mathsf{TSetup}', \mathsf{Prove}', \mathsf{Ver}', \mathsf{Sim}', \mathsf{Ext}')$ *be a NIZK proof of knowledge for the family of languages*

$$L'_{\mathsf{pp}} = \{(\overline{\mathsf{ct}}, \overline{\mathsf{pk}}) \mid \exists(s, r) \text{ such that } s \in \{0,1\}^\lambda \text{ and } \overline{\mathsf{ct}} = \mathsf{HE.Enc}(\overline{\mathsf{pk}}, s; r)\}.$$

Then, the following construction is an HSS supporting the evaluation of polynomial-sized circuits of depth up to $d = d' - d_G$, with one or more inputs in $\{0,1\}^N$ and output in $\{0,1\}$.

- $\mathsf{crs} \leftarrow \mathsf{HSS.Setup}(1^\lambda, 1^{d'})$: *Generate* $\mathsf{pp} \leftarrow \mathsf{HE.Setup}(1^\lambda, 1^{d'})$ *and, for $b = 0, 1$,* $\mathsf{crs}^b_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(\mathsf{pp})$, $\mathsf{crs}^b_{\mathsf{NIZK}'} \leftarrow \mathsf{NIZK}'.\mathsf{Setup}(\mathsf{pp})$. *Output* $\mathsf{crs} := (\mathsf{pp}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$.
- $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{HSS.KeyGen}(b, \mathsf{crs})$: *Parse* $\mathsf{crs} = (\mathsf{pp}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. *Sample* $(\overline{\mathsf{pk}}_b, \overline{\mathsf{sk}}_b) \leftarrow \mathsf{HE.KeyGen}(b, \mathsf{pp}; r)$ *for random r and generate* $\pi_b \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}^b_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_b, (\overline{\mathsf{sk}}_b, r))$. *Set* $\mathsf{pk}_b := (\overline{\mathsf{pk}}_b, \pi_b)$ *and* $\mathsf{sk}_b := \overline{\mathsf{sk}}_b$ *and output* $(\mathsf{pk}_b, \mathsf{sk}_b)$.
- $\mathsf{pk} \leftarrow \mathsf{HSS.KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$: *Parse* $\mathsf{crs} = (\mathsf{pp}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. *For $b \in \{0,1\}$ parse* $\mathsf{pk} = (\overline{\mathsf{pk}}_b, \pi_b)$. *If* $\mathsf{NIZK.Verify}(\mathsf{crs}^0_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_0, \pi_0) = \mathsf{Verify}(\mathsf{crs}^1_{\mathsf{NIZK}}, \overline{\mathsf{pk}}_1, \pi_1) = 1$, *output* $\mathsf{pk} \leftarrow \mathsf{HE.KeyComb}(\mathsf{pp}, \overline{\mathsf{pk}}_0, \overline{\mathsf{pk}}_1)$; *otherwise output* $\perp$.
- $\mathsf{ct} \leftarrow \mathsf{HSS.Enc}(b, \mathsf{crs}, \mathsf{pk}, x)$: *For $x \in \{0,1\}^N$ proceed as follows. Parse* $\mathsf{crs} = (\mathsf{pp}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. *Generate seed $s \leftarrow_R \{0,1\}^\lambda$, encryption* $\overline{\mathsf{ct}} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, s; r)$ *and proof* $\pi \leftarrow \mathsf{NIZK}'.\mathsf{Prove}(\mathsf{crs}^b_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}, \mathsf{pk}), (s, r))$. *Set* $\mathsf{OTP} := G(s) \oplus x$. *Output* $\mathsf{ct} := (\overline{\mathsf{ct}}, \pi, \mathsf{OTP})$.
- $t_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i \in [n]}, f)$: *Parse* $\mathsf{crs} = (\mathsf{pp}, \mathsf{crs}^0_{\mathsf{NIZK}}, \mathsf{crs}^1_{\mathsf{NIZK}}, \mathsf{crs}^0_{\mathsf{NIZK}'}, \mathsf{crs}^1_{\mathsf{NIZK}'})$. *Parse* $\mathsf{ct}_i = (\overline{\mathsf{ct}}_i, \pi_i, \mathsf{OTP}_i)$ *and then compute* $\overline{\mathsf{ct}}' \leftarrow \mathsf{HE.Eval}((\overline{\mathsf{ct}}_1, \ldots, \overline{\mathsf{ct}}_n), f')$, *where*

  $$f'(s_1, \ldots, s_n) := f(G(s_1) \oplus \mathsf{OTP}_1, \ldots, G(s_n) \oplus \mathsf{OTP}_n)$$

  (note that this circuit has at most depth $d' = d + d_G$ as required). If $\mathsf{NIZK}'.\mathsf{Verify}(\mathsf{crs}^{b_i}_{\mathsf{NIZK}'}, (\overline{\mathsf{ct}}_i, \mathsf{pk}), \pi_i) = 1$ *for all $i \in [n]$, compute and output* $t_b := \overline{\mathsf{ct}}' \cdot \mathsf{sk}_b$; *otherwise output* $\perp$.
- $u_b \leftarrow \mathsf{HSS.Reduce}(b, \mathsf{sk}_b, t_b, y)$: *Sample $e_b \leftarrow [-B'', B'']^k$, where $B'' := \lfloor (B' - B)/2 \rfloor$, and output* $u_b := (y, t_b - \frac{q}{p} \cdot y \cdot \mathsf{sk}_b + e_b)$.
- $y \leftarrow \mathsf{HSS.RedComb}(u_0, u_1)$: *Parse $u_b =: (y_b, \tilde{u}_b)$. If $y_0 = y_1$ and $\|\tilde{u}_b + \tilde{u}_{1-b}\|_\infty \leq B'$, output $y := y_0$; otherwise output $\perp$.*

We omit the proof of the following theorem, which is largely identical to the proof of Theorem 4.2. The only additional complication is the presence of the smaller error terms, but our choice of bound parameters B, B', B'' ensures the required properties are met.

Theorem C.2. *Let $\lambda \in \mathbb{N}$. Let N, d', k, p, q, B, B' (depending on λ) be such that $B \cdot \lambda^{\omega(1)} \leq B' < q/(2p)$. Let $G : \{0,1\}^\lambda \to \{0,1\}^N$ be a* PRG *with depth d_G as a boolean circuit. Let* HE *satisfy (d', k, p, q, B)-special nearly linear decryption (as defined in Def. 5.3). Let* NIZK *and* NIZK′ *be non-interactive zero-knowledge proof of knowledge systems for the families of languages L_{pp} and L'_{pp}, respectively, as defined in Construction C.1. Then* HSS *as defined in Construction C.1 satisfies reduced share simulatability and verifiability, as per Definitions 3.5 and 3.6, while supporting the evaluation of functions f computable by polynomial-sized circuits of depth up to $d = d' - d_G$, with one or more inputs in $\{0,1\}^N$ and output in $\{0,1\}$.*

We will now describe in more detail how the GSW encryption scheme [GSW13] can be adapted to fit our framework.

Lemma C.3. *Let $\lambda \in \mathbb{N}$ be a security parameter, $\ell \in \mathbb{N}$, d the depth of a boolean circuit $C\colon \{0,1\}^\ell \to \{0,1\}$, LWE parameters $k = \mathsf{poly}(\lambda, d)$ and $n = k \log q + \omega(\log \lambda)$, B-bounded error distribution $\chi = \chi(\lambda, d)$, moduli $p = \lambda^{\omega(1)}$ and $q = B \cdot p \cdot 2^{\omega(d\lambda \log \lambda)}$. If the* $\mathsf{LWE}_{n-1,q,\chi}$ *assumption holds, then there exists a scheme satisfying (d, k, p, q, B)-special nearly linear decryption as defined in Definition 5.3.*

Proof. This construction, based on the GSW scheme, is defined as follows.

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$: Sample the LWE dimension $n = n(\lambda, d)$, error distribution $\chi = \chi(\lambda, d)$ such that χ is β_χ bounded with $\beta_\chi = \mathsf{poly}(\lambda, d)$, modulus q and modulus p such that $p = 4 \cdot \beta_\chi$ and $8 \cdot \beta_\chi^2 \cdot (m+1)^{d+1} < q$. Sample $m = n \log q + \mathsf{poly}(\lambda)$. Next, uniformly sample a matrix $B \leftarrow_R \mathbb{Z}_q^{(n-1)\times m}$. Output
$$\mathsf{crs} := (n, m, p, q, \chi, B).$$
- $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, b)$: Sample $s_b \leftarrow \chi^{n-1}, e_b \leftarrow \chi^m$ and compute $\mathsf{pk}_b \leftarrow s_b B + e_b \in \mathbb{Z}_q^m$. Output pk_b and $\mathsf{sk}_b := (b, -s_b)$.
- $\mathsf{pk} \leftarrow \mathsf{KeyComb}(\mathsf{pk}_0, \mathsf{pk}_1)$: Output $\mathsf{pk} := \mathsf{pk}_0 + \mathsf{pk}_1 \in \mathbb{Z}_q^m$.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x; r)$: Let $A := \begin{pmatrix} \mathsf{pk} \\ B \end{pmatrix}$. Use r to sample a random matrix $R \leftarrow_R \{0,1\}^{m\times m}$. Output
$$\mathsf{ct} = AR + x \cdot G \in \mathbb{Z}_q^{n\times m}.$$
The gadget matrix is defined to be $G = I_n \otimes \mathbf{g}$, where $\mathbf{g} = (1, 2, \ldots, 2^{\llcorner \log_2 q \lrcorner})$, I_n is the identity matrix of size n, and the operation G^{-1} satisfies the following: $G(G^{-1}(\mathsf{ct})) = \mathsf{ct}$.
- $\mathsf{ct}' \leftarrow \mathsf{Eval}(b, \mathsf{sk}_b, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell, C)$: For any two GSW ciphertexts $\mathsf{ct}_0, \mathsf{ct}_1 \in \mathbb{Z}_q^{n\times m}$, the addition and multiplication are defined as $\mathsf{ct}_0 + \mathsf{ct}_1 \in \mathbb{Z}_q^{n\times m}$ and $\mathsf{ct}_0 \cdot G^{-1}(\mathsf{ct}_1) \in \mathbb{Z}_q^{n\times m}$, respectively. Given the ciphertext ct, the circuit C can be evaluated gate-by-gate. Let the final output ciphertext be $\mathsf{ct}_p \in \mathbb{Z}_q^{n\times m}$. Output $\mathsf{ct}' \leftarrow \mathsf{ct}_p \cdot G^{-1}(\frac{q}{p} \cdot I_n) \in \mathbb{Z}_q^{n\times n}$.
- $\mu \leftarrow \mathsf{Dec}(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{ct}')$: Output the first component of the vector $\left\lceil \frac{p}{q}(\mathsf{sk}_0 + \mathsf{sk}_1) \cdot \mathsf{ct}' \right\rfloor$.

D HSS Instances

Our approach is applicable to relevant 2-party HSS schemes such as the LWE-based construction of [BKS19] and the non-interactive discrete log sharing (NIDLS) framework of [ADOS22] (which notably covers the DCR-based constructions of [OSY21, RS21]).[20] In the two theorems below we observe that these schemes can be easily adapted to satisfy non-interactive key setup and non-adaptive robust linear reconstruction, as in Definitions 3.1 and 5.1, respectively.

Theorem D.1. *Let* HSS *be the LWE-based homomorphic secret sharing scheme [BKS19, Theorem 2]. Under the same assumptions, there exists a secure scheme* HSS′ *with non-interactive key setup and non-adaptive robust linear reconstruction.*

Proof. We begin by informally describing the necessary changes and at the end of the proof we present the scheme HSS′. The keys of the LWE-based HSS scheme are typically of the form $(\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1)$ with $\mathsf{sk} := \mathsf{sk}_0 + \mathsf{sk}_1 = (1, -s)$, $\mathsf{pk} = (As + e, A)$ where $s \leftarrow \mathbb{Z}_q^n$, $e \leftarrow \chi^m$, $A \leftarrow \mathbb{Z}_q^{m\times n}$, and χ is a B-bounded distribution for some norm bound $B \in \mathbb{N}$. To generate these keys non-interactively, we sample the public matrix $A \leftarrow \mathbb{Z}_q^{m\times n}$ as part of the CRS, during the setup step. Then, each party P_b can sample $s_b \leftarrow \mathbb{Z}_q^n$, $e_b \leftarrow \chi^m$ and define $\mathsf{sk}_b := (b, -s_b)$, $\mathsf{pk}_b := As_b + e_b$. The key combination algorithm outputs $\mathsf{pk} := (\mathsf{pk}_0 + \mathsf{pk}_1, A)$. Note that the distribution of the joint public key differs from the public key in the original scheme, as the error term is now $e_0 + e_1$ instead of $e \leftarrow \chi^m$. This does not affect security, and correctness will still hold with overwhelming probability (though concrete parameters may need to be adjusted). We also include in the CRS the common PRF key (used to re-randomize shares during evaluation), which in the original scheme is bundled with the secret keys instead.

We move on to the property of non-adaptive robust linear reconstruction. In the original scheme, the evaluation algorithm computes a vector t_b (a shared representation of the final computation value) and outputs only its first component. All we have to do is modify this algorithm to output the whole vector. Then the secret key has the required format and the condition

$$\mathsf{Eval}(0, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_0, (\mathsf{ct}_i)_i, f) + \mathsf{Eval}(1, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_1, (\mathsf{ct}_i)_i, f) = f(x_1, \ldots, x_n) \cdot \mathsf{sk} \tag{1}$$

is fulfilled with overwhelming probability. For additional clarity, we describe below the adapted scheme HSS′.

[20] All of these constructions support the evaluation of polynomial-size branching programs, which include the class NC1.

- $\mathsf{crs} \leftarrow \mathsf{HSS}'.\mathsf{Setup}(1^\lambda)$: Sample a PRF key k_{PRF} and $A \leftarrow \mathbb{Z}_q^{m\times n}$. Output $\mathsf{crs} := (k_{\mathsf{PRF}}, A)$.
- $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{HSS}'.\mathsf{KeyGen}(b, \mathsf{crs})$: Sample $s_b \leftarrow \mathbb{Z}_q^n$, $e_b \leftarrow \chi^m$ and output $\mathsf{sk}_b := (b, -s_b)$, $\mathsf{pk}_b := As_b + e_b$.
- $\mathsf{pk} \leftarrow \mathsf{HSS}'.\mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$: Output $\mathsf{pk} := (\mathsf{pk}_0 + \mathsf{pk}_1, A)$.
- $\mathsf{ct} \leftarrow \mathsf{HSS}'.\mathsf{Enc}(b, \mathsf{crs}, \mathsf{pk}, x)$: Output $\mathsf{ct} \leftarrow \mathsf{HSS}.\mathsf{Enc}(b, \mathsf{crs}, \mathsf{pk}, x)$.
- $t_b \leftarrow \mathsf{HSS}'.\mathsf{Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$: Let t_b be the representation of the terminal value of the computation, as computed by $\mathsf{HSS}.\mathsf{Eval}$ just before the algorithm halts. Output t_b.

Theorem D.2. *Let* HSS *be the NIDLS homomorphic secret sharing scheme [ADOS22, Theorem 1]. Under the same assumptions, there exists a secure scheme* HSS′ *with non-interactive key setup and non-adaptive robust linear reconstruction.*

Proof. The NIDLS scheme has keys of the form $s_0, s_1 \leftarrow \mathbb{Z}_\ell$ and $\mathsf{pk} := g^{\mathsf{sk}_1 - \mathsf{sk}_0}$, where g is a group generator. These can be generated non-interactively by simply setting $\mathsf{pk}_b := g^{s_b}$ for $b \in \{0,1\}$ and combining them as $\mathsf{pk} := \mathsf{pk}_1 \cdot \mathsf{pk}_0^{-1}$.

In order to satisfy non-adaptive robust linear reconstruction, we can consider an extended secret key $\mathsf{sk} := (1, s)$, and likewise secret key shares $\mathsf{sk}_b := (b, s_b)$ for $b \in \{0,1\}$. During evaluation, the shared representation of an intermediate computation value x is of the form $M_{x,b} = (\langle x\rangle_b, \langle x \cdot s\rangle_b)$, where $\langle\cdot\rangle$ denotes additive shares over the integers (i.e. $\langle x\rangle_0 + \langle x\rangle_1 = x$). At the end, Eval outputs the first component of $M_{y,b}$, where y is the final result of the computation. Notice that we can view memory values as $M_{x,b} = \langle x \cdot \mathsf{sk}\rangle_b$. Similarly to the LWE case, if we define a modified evaluation algorithm which outputs all of $M_{y,b}$, then Equation (1) will be satisfied by sk. The adapted scheme HSS′ consists of the following algorithms.

- $\mathsf{crs} \leftarrow \mathsf{HSS}'.\mathsf{Setup}(1^\lambda)$: Sample a PRF key k_{PRF} and the public parameters $(g, \mathsf{par}) \leftarrow \mathsf{NIDLS}.\mathsf{Setup}(1^\lambda)$. Output $\mathsf{crs} := (k_{\mathsf{PRF}}, g, \mathsf{par})$.
- $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{HSS}'.\mathsf{KeyGen}(b, \mathsf{crs})$: Sample $s_b \leftarrow \mathbb{Z}_\ell$ and output $\mathsf{sk}_b := (b, s_b)$, $\mathsf{pk}_b := g^{s_b}$.
- $\mathsf{pk} \leftarrow \mathsf{HSS}'.\mathsf{KeyComb}(\mathsf{crs}, \mathsf{pk}_0, \mathsf{pk}_1)$: Output $\mathsf{pk} := \mathsf{pk}_1 \cdot \mathsf{pk}_0^{-1}$.
- $\mathsf{ct} \leftarrow \mathsf{HSS}'.\mathsf{Enc}(b, \mathsf{crs}, \mathsf{pk}, x)$: Output $\mathsf{ct} \leftarrow \mathsf{HSS}.\mathsf{Enc}(b, \mathsf{crs}, \mathsf{pk}, x)$.
- $t_b \leftarrow \mathsf{HSS}'.\mathsf{Eval}(b, \mathsf{crs}, \mathsf{pk}, \mathsf{sk}_b, (b_i, \mathsf{ct}_i)_{i\in[n]}, f)$: Let $M_{y,b}$ be the representation of the terminal value of the computation, as computed by $\mathsf{HSS}.\mathsf{Eval}$ just before the algorithm halts. Output $M_{y,b}$.