Verifiable Decryption in the Head

Kristian Gjøsteen[1 [0000−0001−7317−8625]], Thomas Haines[1,2], Johannes Müller[3 [0000−0003−2134−3099]], Peter Rønne[3,4 [0000−0002−2785−8301]], and Tjerand Silde[1 [0000−0002−5455−0409]]

[1] Norwegian University of Science and Technology
{kristian.gjosteen,tjerand.silde}@ntnu.no
[2] Australian National University
thomas.haines@anu.edu.au
[3] University of Luxembourg
johannes.mueller@uni.lu
[4] Université de Lorraine, CNRS, LORIA
peter.roenne@gmail.com

Abstract. In this work we present a new approach to verifiable decryption which converts a 2-party passively secure distributed decryption protocol into a 1-party proof of correct decryption. This leads to an efficient and simple verifiable decryption scheme for lattice-based cryptography, especially for large sets of ciphertexts; it has small size and lightweight computations as we reduce the need of zero-knowledge proofs for each ciphertext. We believe the flexibility of the general technique is interesting and provides attractive trade-offs between complexity and security, in particular for the interactive variant with smaller soundness.
Finally, the protocol requires only very simple operations, making it easy to correctly and securely implement in practice. We suggest concrete parameters for our protocol and give a proof of concept implementation, showing that it is highly practical.

Keywords: verifiable decryption · distributed decryption · lattice-based crypto · MPC-in-the-Head · zero-knowledge proof · implementation

1 Introduction

There are many applications where we not only need to decrypt a ciphertext, but also prove that we have decrypted the ciphertext correctly without revealing the secret key. This is called *verifiable decryption*. Examples include mix-nets used for anonymous communication [42], decryption of ballots in electronic voting [29], and various uses of verifiable fully homomorphic encryption [35]. In particular, such applications usually require the decryption of a large number of ciphertexts.

It is well-known how to do verifiable decryption for public-key encryption schemes based on discrete logarithms (for ElGamal, proving the equality of two discrete logarithms [19] will do). Except for the recent publication by Lyubashevsky *et al.* [38] (which provides a rather complicated decryption proof by combining proofs of linear relations, multiplications and range proofs), no efficient and straight-forward zero-knowledge proofs of correct decryption are known for lattice-based cryptography or other post-quantum encryption schemes. This state-of-affairs is unsatisfying, in particular because many applications that require zero-knowledge proofs of correct decryption should also be secure in the face of quantum computers which are becoming increasingly more powerful. For example, the electronic voting system Helios [1] and the Estonian voting protocol [30] are using classical encryption schemes and decryption proofs with corresponding quantum threats to the long-term privacy of the voters.

On the contrary, there do exist efficient and straightforward passively secure lattice-based encryption schemes with distributed decryption. In such a scheme, the decryption key is shared among several players. Decryption is done in a distributed fashion by each player creating a decryption share, which can be individually verified, and a reconstruction algorithm can recover the message from the decryption shares. *Distributed decryption* allows more general methods to recover the message, such as general multi-party computation. There are many useful and efficient lattice-based threshold cryptosystems and distributed decryption schemes [11,13,16,21,22,24]. In particular, if the security requirements are relaxed, lattice-based distributed decryption can be very straight-forward.

Our main idea is to use MPC-in-the-head [31] in conjunction with a 2-party passively secure distributed decryption scheme to construct a very simple verifiable decryption scheme; however, we shall see that there are various technical challenges. To achieve the desired level of security, we run the 2-party decryption scheme on the ciphertexts many times locally, and then reveal a random subset of keys, one for each run, allowing others to verify that it was done correctly.

1.1 Contribution

Our main contribution is a transformation from a 2-party passively secure distributed decryption scheme to a 1-party verifiable decryption scheme. To achieve this, we use MPC-in-the-head with the 2-party decryption scheme. The idea is that the prover runs the 2-party decryption protocol many times and reveals the resulting decryption shares. The interactive verifier will then, for each run of the decryption scheme, ask to see one of the two decryption keys and any randomness involved in creating the corresponding decryption shares. With this information, it is straight-forward for the verifier to ensure that half of the decryption shares were generated honestly.

As usual, the idea is that if the prover cheats, the verifier will have probability (close to) $1/2$ of detecting this in each round. If a cheating prover is consistently successful, we can use rewinding to extract both secret shares. Furthermore, if the 2-party decryption scheme is passively secure, revealing one share will not reveal anything about the secret key itself.

There are four remaining obstacles, two easy and two somewhat trickier. The first easy obstacle is that in a threshold public key encryption scheme or distributed decryption scheme, the decryption key shares are generated as part of key generation. We already have a decryption key, but we need to create many independent sharings of that key. For discrete logarithm-based schemes like ElGamal, this is usually trivial. For the schemes we consider, it is still not hard, but it follows that we do not have a fully general reduction from 2-party distributed decryption to (1-party) verifiable decryption. The second easy obstacle is that given both secret key shares we want to recover the secret key. We solve this by extending the notation of a distributed decryption function with a function which recovers the key from the shares. This is easy to satisfy in practice.

The third obstacle is that the verifier needs to make sure that the revealed key share is correct. For ordinary threshold decryption schemes, this can often be avoided, either because the dealer is trusted or replaced by some multi-party computation. Therefore, we need to use a non-generic solution here. For batched decryption, the main observation is that we only verify the key once for each run of the 2-party decryption scheme, not once per ciphertext in the batch. The number of runs essentially corresponds to the security parameter, which in many applications will be significantly smaller than the number of ciphertexts.

The final obstacle is related to our security proof. We need to simulate shares of the decryption key, any auxiliary information related to them, and decryption shares. Although similar techniques are common in the construction of threshold public key encryption scheme, the security definitions do not actually require their presence. Since we need them, our approach is again somewhat non-generic.

On the other hand, since we intend to verify correctness of decryption shares by revealing decryption key shares and any randomness involved, we can make do with a passively secure distributed decryption scheme, simplifying our work.

The result is a construction from a somewhat specialized 2-party distributed decryption scheme to a verifiable decryption scheme. Since the security requirements for the distributed decryption scheme are shifted compared to traditional threshold decryption schemes, this will allow us to use very simple threshold decryption. This means that it can be very efficient, both with respect to computational time and size of the decryption shares. Even though the decryption is run many times, the result will still be efficient compared to the alternatives.

Note that in an interactive setting, it may make sense to use a very small security parameter, making the protocol extremely cheap. For instance, in any system where detected cheating will have a significant penalty, rational actors will be deterred by even a small chance of detection. However, when the protocol is made non-interactive, this clearly does not work.

In the full version we prove in the interactive theorem prover Coq [12] a simplified variant of our transform and an ElGamal toy example. Regrettably, we are unable to prove the full transform and the lattice example due to limitations in the interactive theorem prover. Indeed, to our knowledge, no interactive theorem prover exists which provides adequate support. Nevertheless, the proof of the simplified variant increases confidence in the result.

It is worth emphasizing that our protocol is very simple to implement (using Stern-based zero-knowledge proofs [32, 34] to ensure that key-shares are well-formed), lowering the bar for deploying our scheme in practice. We note that lattice-based zero-knowledge proofs in general can be very complicated, involving a combination of proofs of linear relations, proofs of shortness and range proofs, in addition to Gaussian sampling, rejection sampling and optimizations exploiting partially splitting rings and automorphisms [6, 38]. Correctly and securely implementing voting systems using primitives based on discrete logarithms is hard [28], and lattice-based primitives makes it harder. In our protocol we only need to sample uniformly random or short elements in any ring of our choice, and use standard cut-and-choose techniques to open committed values, making it easy to use in practice. Concretely, this means that we are not vulnerable to side-channel attacks against Gaussian sampling [18] or rejection sampling [25].

Combined with the main contribution, this gives us a verifiable decryption scheme for a lattice-based public key encryption scheme that is very efficient when the number of ciphertexts is much larger than the security parameter. The protocol is fast and simple, and the proof size is small. We give concrete parameters and a proof of concept implementation of our protocol in Section 6.

1.2 Related Work

Verifiable decryption for ElGamal can be done by proving the equality of two discrete logarithms [19], and can be batched for significantly improved performance when decrypting many ciphertexts [27, 40].

The "dual" Regev system [39] can be used by making the randomness public. However, this is not zero-knowledge and opens for so-called "tagging-attacks" to de-anonymize users in privacy-preserving applications (e.g., e-voting).

Threshold encryption schemes [23] and distributed decryption schemes are now well-understood, and many constructions exist [11], in particular those related to SPDZ [20, 22, 33]. When only passive security is required, these schemes can be quite efficient. Threshold decryption with active security implies verifiable decryption when the verification of decryption shares is a public operation. The problem is that it is often costly to provide a threshold decryption scheme with active security. Our approach gives away a decryption key share and randomness involved, and it is trivial to verify that the key share has been used correctly.

We compare more in detail with recently developed verifiable decryption protocols [11, 15, 38, 44] in Section 7.

2 Passively Secure 2-party Decryption

A *distributed decryption scheme* enables a set of players to distribute the decryption of ciphertexts, in such a way that only authorized subsets of players can do the decryption. Usually, the decryption key shares are created once during key generation. As discussed in the introduction, we will generate independent decryption key sharings repeatedly, so we need to define the syntax of our variant of distributed decryption schemes precisely.

Consider a public key cryptosystem with key generation algorithm $\mathtt{KeyGen}$, encryption algorithm Enc and decryption algorithm Dec. We extend the notation with a predicate KeyM for key-matching which takes as input a public and secret key. We require for all matching public and secret keys pk, sk and all messages m, that $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$ (with overwhelming probability).

A *distributed decryption protocol* for this public key cryptosystem consists of four algorithms, a *dealer* algorithm, a *verify* algorithm, a *player* algorithm, and a *reconstruction* algorithm. We consider only two parties where both decrypt.

The dealer algorithm (Deal) takes as input a public key and corresponding secret key and outputs two *secret key shares* and some *auxiliary data* aux.

The verify algorithm (Verify) takes as input a public key, auxiliary data, an index and a secret key share and outputs *yes* (1) or *no* (0).

The player algorithm (Play) takes as input a secret key share and a ciphertext and outputs a *decryption share* ds.

The reconstruction algorithm (Rec) takes as input a ciphertext and two decryption shares and outputs either $\perp$ or a message.

Intuitively, the protocol is *correct* if Play and Rec collectively recover the encrypted message and verification accepts when the dealer is honest.

Definition 1 (Correctness). *A distributed decryption protocol is* correct *if for any key pair* $(\mathsf{pk}, \mathsf{sk})$ *s.t.* $\mathsf{KeyM}(\mathsf{pk}, \mathsf{sk}) = 1$*, all* $c = \mathsf{Enc}(\mathsf{pk}, m)$*, any* $(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{aux})$ *output by* $\mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$*, then, for* $i = 0, 1$*,* $\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}, i, \mathsf{sk}_i) = 1$*, and*

$$\mathsf{Pr}[\ m \leftarrow \mathsf{Dec}(\mathsf{sk}, c); \mathsf{Rec}(c, \mathsf{Play}(\mathsf{sk}_0, c), \mathsf{Play}(\mathsf{sk}_1, c)) = m\ ] \geq 1 - \mathsf{negl}.$$

For a distributed decryption protocol, we must trust the dealer for privacy, but not for integrity. The integrity property below says that if both secret shares given by the dealer are valid (according to the Verify algorithm), then the Play and Rec will collectively recover the encrypted message.

Definition 2 (Integrity). *A distributed decryption protocol has* integrity *if there exists an efficient algorithm (named* FindKey *which takes as input the public key, the two secret key shares and the auxiliary information, and returns a secret key) such that for all public keys pk, ciphertexts* $c = \mathsf{Enc}(\mathsf{pk}, m)$*, secret key shares* $(\mathsf{sk}_1, \mathsf{sk}_2)$*, and auxiliary data* aux *and* sk *output by* $\mathsf{FindKey}(\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1, \mathsf{aux})$ *satisfying* $\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}, i, \mathsf{sk}_i) = 1$*, for* $i = 0, 1$*, we have that*

$$\mathsf{Pr}[\ \mathsf{KeyM}(\mathsf{pk}, \mathsf{sk}) \wedge \mathsf{Rec}(c, \mathsf{Play}(\mathsf{sk}_0, c), \mathsf{Play}(\mathsf{sk}_1, c)) = \mathsf{Dec}(\mathsf{sk}, c)\ ] \geq 1 - \mathsf{negl}.$$

For threshold cryptosystems and distributed decryption, security is typically defined through the usual security games for public key cryptosystem, allowing the adversary access to the decryption key shares through decryption share oracles. This security notion is not very convenient for us, so we shall instead rely on a variant of simulatability, namely we must be able to simulate both decryption key shares and decryption shares in a consistent fashion.

$Exp_{\mathcal{A}}^{ddp-sim-0}(\mathsf{pk}, \mathsf{sk})$	$Exp_{\mathcal{A}}^{ddp-sim-1}(\mathsf{pk})$
$(i, (c_0, ..., c_\tau), (m_0, ..., m_\tau)) \leftarrow \mathcal{A}(\mathsf{pk})$	$(i, (c_0, ..., c_\tau), (m_0, ..., m_\tau)) \leftarrow \mathcal{A}(\mathsf{pk})$
$(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{aux}) \leftarrow \mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$	$(\mathsf{sk}_i, \mathsf{aux}) \leftarrow \mathsf{DealSim}(\mathsf{pk}, i)$
$\forall j : \mathsf{ds}_j \leftarrow \mathsf{Play}(\mathsf{sk}_{1-i}, c_j)$	$\forall j : \mathsf{ds}_j \leftarrow \mathsf{PlaySim}(\mathsf{pk}, \mathsf{sk}_i, c_j, m_j)$
$b = \mathcal{A}(\mathsf{aux}, \mathsf{sk}_i, (\mathsf{ds}_0, ..., \mathsf{ds}_\tau))$	$b = \mathcal{A}(\mathsf{aux}, \mathsf{sk}_i, (\mathsf{ds}_0, ..., \mathsf{ds}_\tau))$
return b	**return** b

Fig. 1. The passively secure experiment for distributed decryption protocols.

Definition 3 (Simulatability). *Consider a pair of algorithms* DealSim *and* PlaySim *and an adversary* $\mathcal{A}$ *playing the experiments from Figure 1, where* $\mathcal{A}$ *always outputs* $\boldsymbol{c} = (c_0, ..., c_\tau), \boldsymbol{m} = (m_0, ..., m_\tau)$ *such that* $\{m_j = \mathsf{Dec}(\mathsf{sk}, c_j)\}_{j=1}^{\tau}$. *The* simulatability advantage *of* $\mathcal{A}$ *is*

$$Adv^{ddp-sim}(\mathcal{A}, \mathsf{pk}, \mathsf{sk}) = |\Pr[Exp_{\mathcal{A}}^{ddp-sim-0}(\mathsf{pk}, \mathsf{sk}) = 1] - \Pr[Exp_{\mathcal{A}}^{ddp-sim-1}(\mathsf{pk}) = 1]|,$$

where the probability is taken over the random tapes and $(\mathsf{pk}, \mathsf{sk})$ *output by* `KeyGen`. *We say that a distributed decryption protocol is* (t, ϵ)-simulatable *(or just* simulatable*) if no* t*-time algorithm* $\mathcal{A}$ *has advantage greater than* ϵ.

We give an ElGamal toy example in the full version to showcase our technique.

3 Verifiable Decryption from Distributed Decryption

We will now construct a (batch) zero-knowledge proof system of correct decryption from the distributed decryption protocol. The protocol is given in Figure 2. More precisely, our proof system is a sigma protocol with completeness, special soundness, and honest verifier-zero knowledge.

For any public key cryptosystem, a public key output by the key generation algorithm uniquely defines a decryption function that for all messages agrees with the decryption algorithm for any ciphertext output by the encryption algorithm, except those that lead to decryption failure.

Recall that for a batched verifiable decryption protocol the statement consists of a public key, a vector of ciphertexts and a vector of messages, where the ciphertexts have been output by the encryption algorithm. The statement is in the language if and only if the messages correspond to the decryption function applied to the ciphertexts. The secret key (witness) satisfies the relationship with the statement if it corresponds to the public key and the message vector is the decryption of the ciphertexts with the secret key.

The protocol works as follows: the prover creates λ sharings of the secret key by calling the Deal algorithm λ times. For each sharing and each ciphertext, the prover uses the Play algorithm to construct the decryption share. The prover sends the auxiliary information from Deal and all the shares to the verifier. Then, the verifier returns a challenge which is a binary vector of length λ. The prover finally reveals the corresponding parts of the shares as well as any randomness used in the Play algorithms with this key share. The prover checks that (1) all the revealed shares verify, (2) the decryption shares are consistent with the revealed key shares, and (3) the messages correspond to the decryption shares.

Π_{ZKPCD}

Prover$((\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau}); (\mathsf{sk}))$		Verifier$(\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau})$
$k = 1, ..., \lambda:$		
$\quad(\mathsf{sk}_{0,k}, \mathsf{sk}_{1,k}, \mathsf{aux}_k) \leftarrow \mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$		
$\quad i = 0, 1:$		
$\quad\quad j = 1, ..., \tau:$		
$\quad\quad\quad \mathsf{ds}_{i,j,k} \leftarrow \mathsf{Play}(\mathsf{sk}_{i,k}, c_j; \rho_{i,k,j})$		
$w \leftarrow (\{\mathsf{aux}_k, \{t_{i,j,k}\}\})$		
	$\xrightarrow{w}$	
		$\boldsymbol{\beta} \leftarrow\$ \{0,1\}^{\lambda}$
	$\xleftarrow{\boldsymbol{\beta}}$	
$z \leftarrow (\{\mathsf{sk}_{\boldsymbol{\beta}[k],k}\}_k, \{\rho_{\boldsymbol{\beta}[k],k,j}\}_{k,j})$		
	$\xrightarrow{z}$	
		$k = 1, ..., \lambda:$
		$\quad\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}_k, \boldsymbol{\beta}[k], \mathsf{sk}_{\boldsymbol{\beta}[k],k}) \stackrel{?}{=} 1$
		$\quad j = 1, ..., \tau:$
		$\quad\quad\mathsf{Play}(\mathsf{sk}_{\boldsymbol{\beta}[k],k}, c_j; \rho_{\boldsymbol{\beta}[k],k,j}) \stackrel{?}{=} \mathsf{ds}_{\boldsymbol{\beta}[k],j,k}$
		$\quad\quad\mathsf{Rec}(c_j, \mathsf{ds}_{0,j,k}, \mathsf{ds}_{1,j,k}) \stackrel{?}{=} m_j$

Fig. 2. Proof of correct decryption. $\rho_{i,k,j}$ denotes the random tape used by the Play algorithm to create the ith share of the jth ciphertext in the kth run of the protocol.

Completeness. Up to the possible negligible error introduced by decryption failures, completeness follows immediately by construction and the correctness of the underlying distributed decryption protocol.

Special Soundness. By rewinding, any cheating prover with a significant success probability can be used to create two accepting conversations $(w, \boldsymbol{\beta}, z)$ and

$(w, \boldsymbol{\beta}', z')$, with $\boldsymbol{\beta} \neq \boldsymbol{\beta}'$. From this it follows that $\boldsymbol{\beta}[k] \neq \boldsymbol{\beta}'[k]$ for at least one k, and the verify algorithm has accepted both secret key shares and every decryption share in this round has been correctly created using the Play algorithm. Then, since the ciphertexts are encryptions of the first message vector, integrity implies that $\mathsf{FindKey}$ will recover a witness which matches the public key and for which the messages match the output of the decryption function.

Honest-Verifier Zero-Knowledge. Our simulator works as follows, given the statement $(\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau})$ and the challenge $\boldsymbol{\beta}$: First, for $i = 1, ..., \lambda$, we let $(\mathsf{aux}_i, \mathsf{sk}_{\boldsymbol{\beta}[i],i}) \leftarrow \mathsf{DealSim}(\mathsf{pk}, \boldsymbol{\beta}[i])$ and, for $j = 1, ..., \tau$, we let $\mathsf{ds}_{\boldsymbol{\beta}[i],j,i} \leftarrow \mathsf{PlaySim}(\mathsf{pk}, \mathsf{sk}_{\boldsymbol{\beta}[i],i}, c_i, m_i)$ and $\mathsf{ds}_{1-\boldsymbol{\beta}[i],j,i} \leftarrow \mathsf{Play}(\mathsf{pk}, \mathsf{sk}_{\boldsymbol{\beta}[i],i}, c_i)$. The proof transcripts is then $((\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau}), (\mathsf{aux}_i, \mathsf{ds}_{0,j,i}, \mathsf{ds}_{1,j,i}), \boldsymbol{\beta}, \mathsf{sk}_{\boldsymbol{\beta}[i],i})$. This is computationally indistinguishable from the honest transcripts if the distributed decryption protocol is simulatable.

We give a machine checked proof of our protocol instantiated with ElGamal in the full version of this paper to provide confidence in our general transform.

4 BGV Encryption

We present a version of the BGV encryption scheme by Brakerski, Gentry and Vaikuntanathan [17]. See the full version of this paper for background on lattice-based cryptography. Let $p \ll q$ be primes, let R_q and R_p be polynomial rings modulo the primes q or p and $X^N + 1$ for a fixed N, let $B_\infty \in \mathbb{N}$ be a bound and let κ be the security parameter. The encryption scheme consists of three algorithms: key generation, encryption and decryption, where

- `KeyGen` samples an element $a \leftarrow\$ R_q$ uniformly at random, samples short $s, e \leftarrow\$ R_q$ such that $\max(\|s\|_\infty, \|e\|_\infty) \leq B_\infty$. The algorithm outputs the public key $\texttt{pk} = (a, b) = (a, as + pe)$ and the secret key $\texttt{sk} = (s, e)$.
- `Enc`, on input the public key $\texttt{pk} = (a, b)$ and an element m in R_p, samples short $r, e', e'' \leftarrow\$ R_q$ such that the norm $\max(\|r\|_\infty, \|e'\|_\infty, \|e''\|_\infty) \leq B_\infty$, and outputs the ciphertext $c = (u, v) = (ar + pe', br + pe'' + m)$ in R_q^2.
- `Dec`, on input the secret key $\texttt{sk} = (s, e)$ and a ciphertext $c = (u, v)$, outputs the message $m = (v - su \mod q) \mod p$ in R_p.

The decryption algorithm is correct as long as the norm $\max\|v - su\|_\infty = B_{\texttt{Dec}} < \lfloor q/2 \rceil$. It follows that the BGV encryption scheme is secure against chosen plaintext attacks if the $\mathsf{DKS}^{\infty}_{N,q,\beta}$ problem is hard for some $\beta = \beta(N, q, p, B_\infty)$.

Furthermore, we present the passively secure distributed decryption technique by Bendlin and Damgård [11] used in the MPC-protocols by Damgård *et al.* [20, 22]. When decrypting, we assume that each decryption server $\mathcal{D}_j$, for $1 \leq j \leq \xi$, has a uniformly random share $\mathsf{sk}_j = s_j$ of the secret key $\mathsf{sk} = (s, e)$ such that $s = s_1 + s_2 + ... + s_\xi$. Then they partially decrypt in the following way:

- `DistDec`, on input a secret key-share $\mathsf{sk}_j = s_j$ and a ciphertext $c = (u, v)$, computes $m_j = s_j u$, sample some large noise $E_j \leftarrow\$ \mathbb{E} \subset R_q$ such that $\|E_j\|_\infty \leq 2^{\texttt{sec}}(B_{\texttt{Dec}}/p\xi)$ for some statistical security parameter `sec` and upper error-bound $\max\|v - su\|_\infty \leq B_{\texttt{Dec}}$, then outputs $\mathsf{ds}_j = t_j = m_j + pE_j$.

We obtain the full decryption of the ciphertext (u, v) as $m \equiv (v - t \mod q) \mod p$, where $t = t_1 + t_2 + ... + t_\xi$. This will give the correct decryption as long as the noise $\max\|v - t\|_\infty \leq (1+2^{\mathsf{sec}})B_{\mathsf{Dec}} < \lfloor q/2 \rfloor$ (see [20, Appendix G]). Here, t will be indistinguishable from random except with probability $2^{-\mathsf{sec}}$.

5 Zero-Knowledge Protocol of Correct Decryption

5.1 Lattice-Based Distributed Decryption

Setup. We will be working over the ring $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$ together with a modulus $p \ll q$, both prime. These are the public parameters of the protocol, together with security parameter κ, soundness parameter λ, bound B_∞ and maximal ciphertext error-bound B_{Dec}. We define commitments, their security and give a concrete instantiation based on lattices in the full version of this paper. The commitments are both computationally hiding and computationally binding, in addition to being linearly homomorphic. Finally, let $(\Pi_{\mathrm{ZKPoS}}, \Pi_{\mathrm{ZKPoSV}})$ be a non-interactive zero-knowledge protocol for the following relation:

$$R_{\mathsf{DKS}^\infty_{N,q,1}} = \{((\boldsymbol{A}, \boldsymbol{y}); \boldsymbol{x}) \colon \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \mod q \wedge \|\boldsymbol{x}\|_\infty = 1\}.$$

Scheme. We present a distributed decryption version of the BGV encryption scheme [17], where $\mathtt{KeyGen}$, Enc and Dec are defined in Section 4.

The dealer algorithm (Deal) takes as input a public key $\mathsf{pk} = (a, b)$ and corresponding secret key $\mathsf{sk} = (s, e)$, samples uniform s_0 and e_0 from R_q, and computes $s_1 = s - s_0$ and $e_1 = e - e_0$. Then it commits to the values as $c_{s_i} = \mathtt{Com}(s_i), c_{e_i} = \mathtt{Com}(e_i)$, and computes $b_i = as_i + pe_i$ so that $b = b_0 + b_1$. Finally, it computes non-interactive zero-knowledge proofs π_{S_i} proving that the sums $s_0 + s_1$ and $e_0 + e_1$ are short (see details in Section 6). It outputs key shares $sk_0 = (s_0, e_0), sk_1 = (s_1, e_1)$ and $\mathsf{aux} = (b_0, b_1, c_{s_0}, c_{s_1}, c_{e_0}, c_{e_1}, \pi_{S_0}, \pi_{S_1})$.

The verify algorithm (Verify) takes as input a public key $\mathsf{pk} = (a, b)$, an index i, a secret key share $\mathsf{sk}_i = (s_i, e_i)$, openings d_{s_i} and d_{e_i}, and aux. It outputs 1 if and only if $(b_i \stackrel{?}{=} as_i + pe_i) \wedge (b \stackrel{?}{=} b_0 + b_1) \wedge \mathtt{Open}(c_{s_i}, d_{s_i}) \wedge \mathtt{Open}(c_{e_i}, d_{e_i}) \wedge (\Pi_{\mathrm{ZKPoSV}}(\mathsf{sk}_i, \mathsf{aux}, \pi_{S_i}))$, and 0 otherwise.

The player algorithm (Play) takes as input a key share $\mathsf{sk}_i = (s_i, e_i)$, a ciphertext $c = (u, v)$, samples bounded E_i and outputs $\mathsf{ds}_j = t_i = s_i u + pE_i$.

The reconstruction algorithm (Rec) takes as input a ciphertext $c = (u, v)$, decryption shares (t_0, t_1), and outputs $m = (v - t_0 - t_1 \mod q) \mod p$.

5.2 Security

Theorem 1 (Correctness). *The distributed decryption scheme in 5.1 is correct with respect to Definition 1 when* $\max\|v - t\|_\infty \leq (1 + 2^{\mathsf{sec}})B_{\mathsf{Dec}} < \lfloor q/2 \rfloor$.

Theorem 2 (Integrity). *Suppose the protocol Π_{ZKPoS} is (computationally) sound and that* `Com` *is (computationally) binding. Let $\mathcal{A}_0$ be an adversary against integrity of the distributed decryption scheme with advantage ϵ_0, and let λ be the number of rounds in the protocol. Then there exists adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ against soundness of Π_{ZKPoS} and binding of* `Com`*, respectively, with advantages ϵ_1 and ϵ_2, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + 2^{-\lambda}$. The runtime of $\mathcal{A}_1$ and $\mathcal{A}_2$ are essentially the same as the runtime of $\mathcal{A}_0$.*

Proof. We sketch the argument. There are essentially three possible ways to attack the integrity of the protocol: an attacker that knows the secret decryption key but correctly guess the challenge in each round is able to decrypt to arbitrary messages, and otherwise, if the attacker does not know the secret key, needs to break the underlying schemes. The guessing attack has success probability $2^{-\lambda}$.

For Verify to accept for both $i = 0$ and $i = 1$, we need that $b = b_0 + b_1$, $b_0 = as_0 + pe_0$, $b_1 = as_1 + pe_1$ and that the zero-knowledge proof of shortness π_S of the sums $s_0 + s_1$ and $e_0 + e_1$ are accepted. If either of the key shares are incorrect then Verify accept with probability 0, and if the key shares are correct, then Rec outputs m except with negligible probability. An attacker can choose s_0, s_1, e_0 and e_1 such that all equations are correct, but the sums are not short. The soundness of Verify then reduces to the soundness of the zero-knowledge protocol, and an attacker $\mathcal{A}_0$ against this part of the protocol with advantage ϵ_0 can be turned into an attacker $\mathcal{A}_1$ against Π_{ZKPoS} with the same advantage.

The last option is for the attacker to produce commitments to a true but unrelated statement with respect to the secret key used in the encryption scheme. This allows the attacker to produce a valid proof of shortness without cheating, but for an unrelated key. However, Verify only accepts if both the opening of the commitments are correct and the zero-knowledge proof of shortness verifies. Hence, and attacker $\mathcal{A}_0$ that is able to produce valid openings and proofs with advantage ϵ_0 can be turned into an attacker $\mathcal{A}_2$ against `Com` with the same advantage by rewinding the prover for the zero-knowledge proof of knowledge of short openings and then extract two different but valid openings to the commitment.

Theorem 3 (Privacy). *Suppose the protocol Π_{ZKPoS} is (statistically) honest-verifier zero-knowledge, that* `Com` *is (computationally) hiding and that* Enc *is (computationally) CPA secure. Then there exists a simulator for the verifiable decryption protocol such that for any distinguisher $\mathcal{A}_0$ for this simulator with advantage ϵ_0 there exists an adversary $\mathcal{A}_2$ against hiding for the commitment scheme with advantage ϵ_2, an adversary $\mathcal{A}_3$ against CPA security for the encryption scheme with advantage ϵ_3, and a distinguisher $\mathcal{A}_1$ for the simulator of Π_{ZKPoS} with advantage ϵ_1, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + \epsilon_3$. The runtime of $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are essentially the same as the runtime of $\mathcal{A}_0$.*

Proof. Let $\texttt{Sim}_{\text{Short}}$ be a simulator for Π_{ZKPoS}. We present a simulator DealSim for the Deal-algorithm and a simulator PlaySim for the Play-algorithm in Figure 3.

DealSim: We create the simulator in three steps. We first replace π_S by the simulated proof π_S^* produced by $\texttt{Sim}_{\text{Short}}$. An attacker $\mathcal{A}_0$ with advantage ϵ_0

$\mathsf{DealSim}(\mathsf{pk} = (a, b), i)$

$i = 0, 1\colon s_i^* \leftarrow\$ R_q, \quad e_i^* \leftarrow\$ R_q$
$b_i^* = as_i^* + pe_i^*, \quad b_{1-i}^* = b - b_i^*$
$c_{s_i}^* \leftarrow \mathtt{Com}(s_i^*), c_{s_{1-i}}^* \leftarrow \mathtt{Com}(s_{1-i})$
$c_{e_i}^* \leftarrow \mathtt{Com}(e_i^*), c_{e_{1-i}}^* \leftarrow \mathtt{Com}(s_{1-i})$
$\pi_S^* \leftarrow \mathtt{Sim}_{\text{Short}}(c_{s_i}^*, c_{s_{1-i}}^*, c_{e_i}^*, c_{e_{1-i}}^*)$
$\mathsf{aux}^* \leftarrow (b_0^*, b_1^*, c_{s_0}^*, c_{s_1}^*, c_{e_0}^*, c_{e_1}^*, \pi_S^*)$
return $(\mathsf{sk}_i^* = (s_i^*, e_i^*), \mathsf{aux}^*)$

$\mathsf{PlaySim}(\mathsf{sk}_{1-i} = (s_{1-i}, e_{1-i}), c = (u, v), i, m)$

$E_{1-i} \leftarrow\$ \mathbb{E}$
$t_{1-i} = s_{1-i}u + pE_{1-i}$
$t_i^* = v - m - t_{1-i} \mod p$
return $(\mathsf{ds}_i^* = t_i^*)$

Fig. 3. Simulators DealSim and PlaySim.

against this change can be turned into an attacker $\mathcal{A}_1$ against the simulator $\mathtt{Sim}_{\text{Short}}$ of protocol Π_{ZKPoS} with the same advantage.

Next, we replace the key shares by uniformly random key-shares s_i^* and e_i^* that give correctness, that is, the public key-shares b_0^* and b_1^* sum to b, but s_0^* and s_1^* does not need to sum to a short key s^* and e_0^* and e_1^* does not need to sum to short noise e^*. This ensures that Verify outputs 1. An attacker $\mathcal{A}_0$ with advantage ϵ_0 against this change can then be turned into an attacker $\mathcal{A}_3$ against CPA security of the encryption scheme with the same advantage.

Finally, we replace the commitments to unopened values by commitments to random values. This way, none of the values in the protocol any longer depends on the secret key in the protocol, and b_i^* are simulated perfectly. An attacker $\mathcal{A}_0$ with advantage ϵ_0 against this change can then be turned into an attacker $\mathcal{A}_2$ against hiding of the commitment scheme with the same advantage.

PlaySim: we start by sampling bounded E_{1-i} from $\mathbb{E}$ and computing $t_{1-i} = s_{1-i}u + pE_{1-i}$. Then we find t_i such that $(v - t_0 - t_1 \mod q) \mod p = m$. This ensures that Rec outputs the message m when reconstructing the shares. Here, the values are sampled according to the exact same distribution as in the real protocol, and the statistical distance is negligible in the security parameter κ.

5.3 Zero-Knowledge Proof of Verifiable Decryption

We present the different phases of our sigma protocol for proving correct decryption. The protocol is given in Figure 4. The security of the construction follows directly from the results in Section 3 in combination with Theorem 1, 2 and 3.

Setup. We are given a honestly generated public key $\mathsf{pk} = (a, b = as + pe)$, where $\max(\|s\|_\infty, \|e\|_\infty) \leq B_\infty$. The secret key $\mathsf{sk} = (s, e)$ is given to the prover. We are given a set of honestly generated ciphertexts $\{(u_j, v_j) = (ar_j + pe'_j, br_j + pe''_j + m_j)\}_{j=1}^{\tau}$, where $\max(\|r\|_\infty, \|e'\|_\infty, \|e''\|_\infty) \leq B_\infty$, and set of messages $\{m_j\}_{j=1}^{\tau}$.

Commit phase. For soundness parameter λ, the prover does the following for $k = 1, ..., \lambda$. First, it runs the Deal algorithm on sk and pk to produce $\mathsf{sk}_{0,k}, \mathsf{sk}_{1,k}$

and aux_k. It uses Π_{ZKPoS} to prove that the shares are correctly computed. Then, for $i = 0, 1$ and each $j = 1, ..., \tau$, it runs the Play algorithm on each key-share $\mathsf{sk}_{i,k}$ and ciphertext c_j to produce $t_{0,j,k}$ and $t_{1,j,k}$. Finally, it sends $w \leftarrow (\{\mathsf{aux}_k, \{t_{i,j,k}\}_{i=0,j=1}^{1,\tau}\}_{k=1}^{\lambda})$ to end the commitment phase.

Challenge phase. The verifier independently samples a random binary challenge vector $\boldsymbol{\beta}$ of length λ. It sends $\boldsymbol{\beta}$ to the prover.

Respond phase. The prover sends openings $z = (\{d_{s_{\boldsymbol{\beta}[k],k}}, d_{e_{\boldsymbol{\beta}[k],k}}\})$, for each of the commitments to each index k of $\boldsymbol{\beta}$, to the verifier.

Verification phase. For each $k = 1, ..., \lambda$, the verifier runs the Verify algorithm to make sure that the openings of $s_{\boldsymbol{\beta}[k],k}$ and $e_{\boldsymbol{\beta}[k],k}$ are valid, check that all shares of the public key are computed correctly as $b_{\boldsymbol{\beta}[k],k} = as_{\boldsymbol{\beta}[k],k} + pe_{\boldsymbol{\beta}[k],k}$, verify the public key $b = b_{0,k} + b_{1,k}$ and ensure that each $\pi_{S_{i,k}}$ is valid. Further, for each $j = 1, ..., \tau$, the verifier runs the Rec algorithm to make sure that all decryption shares are correct and that all messages are decrypted correctly. It outputs 1 if all checks hold, and 0 otherwise.

Fiat-Shamir. To make the scheme non-interactive we can use the Fiat-Shamir transform [26] by hashing the output of the commit phase and use the hash as challenge, before outputting the response. We note that this can be done similarly to the optimizations described for estimating the size in the next section. We also note that the soundness parameter λ initially can be very small in the interactive case, while it should be (approximately) as large at the security parameter κ in the non-interactive setting, increasing the size of the proof of decryption.

Hybrid proof. We note that the interaction in the protocol opens for a hybrid proof: if we wish for a quick result to get confidence in the decrypted ciphertexts but at the same time can wait longer to be completely certain, we can ask for two proofs. First, we ask the prover for a proof where $\lambda_I = 10$ or $\lambda_I = 20$, and sample a random challenge ourselves. If we accept the proof, we ask the prover to compute a non-interactive proof for the same statement but with $\lambda_N = 100$. This proof can be received, stored and verified later, knowing already that the messages most likely are correctly decrypted. The interactive proof also allows the verifier to arbitrarily increase λ_I by sending more challenges on the fly, where we tell the prover when we are done, and he creates the proofs of shortness in the end. This is particularly useful in real-world applications, e.g., e-voting.

6 Performance

In this section, we shall carefully analyze the performance of our decryption proof. Along the way, we make several easy optimizations with respect to the protocol in Fig. 4. In particular, we use a commitment in the first message, and then send only the values that the verifier cannot recompute himself in the

Π_{ZKPCD}

Prover$(((a,b),\{(u_j,v_j)\}_{j=1}^{\tau},\{m_j\}_{j=1}^{\tau});(s,e))$

Verifier$((a,b),\{(u_j,v_j)\}_{j=1}^{\tau},\{m_j\}_{j=1}^{\tau})$

Prover:

$k=1,\ldots,\lambda:$

Deal:

$(s_{0,k},s_{1,k}) \leftarrow\$ \langle s\rangle$

$(e_{0,k},e_{1,k}) \leftarrow\$ \langle e\rangle$

$i=0,1:$

$(c_{s_{i,k}},d_{s_{i,k}}) \leftarrow \mathtt{Com}(s_{i,k})$

$(c_{e_{i,k}},d_{e_{i,k}}) \leftarrow \mathtt{Com}(e_{i,k})$

$b_{i,k} \leftarrow as_{i,k}+pe_{i,k}$

$j=1,\ldots,\tau:$

Play:

$E_{i,j,k} \leftarrow\$ \mathbb{E}$

$t_{i,j,k}=s_iu_j+pE_{i,j,k}$

$\pi_{S_{0,k}} \leftarrow \Pi_{\mathrm{ZKPoS}}(c_{s_{0,k}},c_{s_{1,k}});(d_{s_{0,k}},d_{s_{1,k}}))$

$\pi_{S_{1,k}} \leftarrow \Pi_{\mathrm{ZKPoS}}(c_{e_{0,k}},c_{e_{1,k}});(d_{e_{0,k}},d_{e_{1,k}}))$

$w \leftarrow (\{b_{i,k},c_{s_{i,k}},c_{e_{i,k}},\pi_{S_{i,k}},\{t_{i,j,k}\}_j\}_{i,k})$

Prover $\xrightarrow{\;w\;}$ Verifier

Verifier: $\boldsymbol{\beta} \leftarrow\$ \{0,1\}^{\lambda}$

Prover $\xleftarrow{\;\boldsymbol{\beta}\;}$ Verifier

Prover: $z \leftarrow (\{d_{s_{\boldsymbol{\beta}[k],k}},d_{e_{\boldsymbol{\beta}[k],k}}\}_k)$

Prover $\xrightarrow{\;z\;}$ Verifier

Verifier:

$k=1,\ldots,\lambda:$

Verify:

$\mathtt{Open}(c_{s_{\boldsymbol{\beta}[k],k}},d_{s_{\boldsymbol{\beta}[k],k}}) \stackrel{?}{=} 1$

$\mathtt{Open}(c_{e_{\boldsymbol{\beta}[k],k}},d_{e_{\boldsymbol{\beta}[k],k}}) \stackrel{?}{=} 1$

$1 \stackrel{?}{\leftarrow} \Pi_{\mathrm{ZKPoSV}}(c_{s_{0,k}},c_{s_{1,k}},\pi_{S_{0,k}})$

$1 \stackrel{?}{\leftarrow} \Pi_{\mathrm{ZKPoSV}}(c_{e_{0,k}},c_{e_{1,k}},\pi_{S_{1,k}})$

$b_{\boldsymbol{\beta}[k],k} \stackrel{?}{=} as_{\boldsymbol{\beta}[k],k}+pe_{\boldsymbol{\beta}[k],k}$

$b \stackrel{?}{=} b_{0,k}+b_{1,k}$

$j=1,\ldots,\tau:$

Rec:

$pE_{\boldsymbol{\beta}[k],j,k}=t_{\boldsymbol{\beta}[k],j,k}-u_js_{\boldsymbol{\beta}[k],k}$

$\|pE_{\boldsymbol{\beta}[k],j,k}\|_{\infty} \stackrel{?}{\leq} 2^{\mathsf{sec}-1}B_{\mathtt{Dec}}$

$v_j-t_{0,j,k}-t_{1,j,k} \stackrel{?}{\equiv}_p m_j$

Fig. 4. Zero-knowledge proof of correct decryption.

second message. Finally, we compute the zero-knowledge proofs of shortness in the response phase instead of the commit phase, reducing the number of proofs by a factor of two in each round of the protocol.

6.1 Proof Size

Each element in R_q is of size $N \log q$ bits, which might be large, and each element in R_p is of size $N \log p$ bits, which will be small. Short elements bounded by B_∞ is of size $N \log B_\infty$ bits. We let $\mathtt{H}$ be a collision resistant hash-function with output of length 2κ. Note that the soundness parameter λ may be chosen independently of, and in particular smaller than, the security parameter κ.

Commit phase. To reduce the number of ring elements being sent, we commit to the output of the commit phase using a hash-function, and send the hash instead. More concretely, we let $w = \mathtt{H}(\{b_{0,k}, b_{1,k}, c_{s_{0,k}}, c_{s_{1,k}}, c_{e_{0,k}}, c_{e_{1,k}}, \{t_{i,j,k}\}_{i=0,j=1}^{1,\tau}\}_{k=1}^{\lambda})$.

Challenge phase. The verifier sends the vector $\boldsymbol{\beta}$ consisting of λ independently sampled bits to the prover.

Respond phase. Note that we do not need to send the partial decryptions $t_{\boldsymbol{\beta}[k],j,k}$, because they can be computed uniquely from u_j, $s_{\boldsymbol{\beta}[k],k}$ and $E_{\boldsymbol{\beta}[k],j,k}$, and we can let a uniform binary seed $\rho_{\boldsymbol{\beta}[k],k}$ of length 2κ bits can be used to deterministically generate the randomness used in each round. Next, we also note that $b_{\boldsymbol{\beta}[k],k}$ can be computed directly from $s_{\boldsymbol{\beta}[k],k}$ and $e_{\boldsymbol{\beta}[k],k}$, and $b_{1-\boldsymbol{\beta}[k],k}$ from b and $b_{\boldsymbol{\beta}[k],k}$.

It follows that, for each $k = 1, ..., \lambda$, the prover sends $s_{\boldsymbol{\beta}[k],k}$ and $e_{\boldsymbol{\beta}[k],k}$, commitments $c_{s_{1-\boldsymbol{\beta}[k],k}}$ and $c_{e_{1-\boldsymbol{\beta}[k],k}}$ together with the openings $d_{s_{\boldsymbol{\beta}[k],k}}$ and $d_{e_{\boldsymbol{\beta}[k],k}}$, and the partial decryptions $\{t_{1-\boldsymbol{\beta}[k],j,k}\}_{j=1}^{\tau}$. Since the commitments to the sharings of s and e are used in the zero-knowledge proof of shortness, these commitment is computed using lattice-based commitments. We observe that $c_{s_k} = c_{s_{1-\boldsymbol{\beta}[k],k}} + \mathtt{Com}(s_{\boldsymbol{\beta}[k],k})$ and $c_{e_k} = c_{e_{1-\boldsymbol{\beta}[k],k}} + \mathtt{Com}(e_{\boldsymbol{\beta}[k],k})$, with randomness zero, are commitments to $s_{\boldsymbol{\beta}[k],k} + s_{1-\boldsymbol{\beta}[k],k}$ and $e_{\boldsymbol{\beta}[k],k} + e_{1-\boldsymbol{\beta}[k],k}$, which are short. Then we use the zero-knowledge proof of shortness to prove that we know openings of c_{s_k} and c_{e_k} to get π_{S_0} and π_{S_1}. Denote all proofs of shortness by π_S.

Total communication. The total proof size sent by the prover is

$$2\kappa + \lambda N(4 \log q + 2\kappa + 2 \log B_\infty) + \lambda \tau N \log q + |\pi_S| \text{ bits.}$$

Zero-knowledge proof of shortness. There are many options for π_S, proving knowledge of valid openings of the commitments c_{s_k} and c_{e_k}. We can use the Fiat-Shamir with aborts framework [36,37], but this would give us a large soundness slack, that is, we prove knowledge of a vector that might be much larger than what we started with. This would increase the parameters to be used in the overall protocol. Other alternatives are the exact proofs using MPC-in-the-head techniques by Baum and Nof [9] or the range proofs by Attema *et al.* [6]. However, we note that even though these are efficient, both protocols are very complex and are complicated to implement correctly for use in the real world. Another approach is to use generic proof systems like Ligero [4] or Aurora [10], adding more complexity to the overall protocol. We can also use the amortized proof by Bootle *et al.* [7] to prove that all λ executions are done correctly at the same time. This is the most efficient proof system for these relations today.

However, assuming that the soundness parameter λ is much smaller than the number of ciphertexts τ, the size of the proofs of shortness does not matter much. To keep the protocol as simple as possible, to make it easier to implement the protocol and avoid bugs in practice, we choose to use the Stern-based proofs by Kawachi *et al.* [32] and Ling *et al.* [34] in our implementation and estimates.

Concrete parameters. For a concrete instantiation, we use the example parameters in Table 1, estimated to $\kappa = 128$ bits of long-term security using the LWE-estimator [3] with the *BKZ.qsieve* cost-model. Inserting these parameters into the proof of shortness, then each proof $\pi_{S_{i,k}}$ is of size $\approx 87\mu$ KB. This makes $|\pi_S| \approx 175\mu\lambda$ KB. Furthermore, using the improvements by Beullens [14] we can shrink the proofs down to $18\mu\lambda$ KB. If we replace π_S with the amortized proof by Bootle *et al.* [7] we get a proof of total size 520 KB*. However, if the number of ciphertexts τ is very large, we can ignore all other terms and get a proof of correct decryption π_D of size $\approx 14\lambda\tau$ KB. See Table 1 for details. The given ciphertext modulus q is chosen to be large enough to ensure correct decryption.

Parameter	Explanation	Constraints	Value
N	Dimension	Power of two	2048
q	Ciphertext modulus	$B_{\mathtt{Dec}} \ll q \equiv 1 \mod 2N$	$\approx 2^{55}$
p	Plaintext modulus		2
κ	Security parameter	Long-term privacy	128
$\mathtt{sec}$	Statistical security		40
λ	Soundness parameter		$10, ..., 128$
μ	Repetitions of Π_{ZKPoS}	$\mu \geq \lambda \cdot \ln(2)/\ln(3/2)$	$17, ..., 218$
B_∞	Bounds on secrets		1
$B_{\mathtt{Dec}}$	Decryption bound	$\|v - su\|_\infty \leq B_{\mathtt{Dec}}$	$\approx 2^{13}$
Size of π_D	Timings for π_D	Size of π_S	Timings for π_S
$14\lambda\tau$ KB	$4\lambda\tau$ ms	$175\lambda\mu$ KB	$30\lambda\mu$ ms

Table 1. Notation, explanation, constraints and concrete parameters for the protocol. We also provide size and timings for decryption proof π_D and proofs of shortness π_S.

* Setting $m = 2048, \log q = 55, r = 90, b = 3, \tau = 50, k = 2398, l = 5000$ and $h = 100$ for soundness 2^{-45} and run the protocol twice, see [7, Section 4.1] for details.

6.2 Implementation

We wrote a proof of concept implementation of our scheme in C++ using the NTL-library [43]. The implementation was benchmarked on an Intel Core i5 running at 2.3 GHz with 16 GB RAM. We ran the protocol with $\lambda = 40, \tau = 1000, \mu = 68$. The timings are given in Table 1. The implementation is very simple, and consists of a total of 400 lines of code. Our source code is available online **. We note that our implementation does not use the number theoretic transform for fast multiplication of elements in the ring to reduce complexity. A rough comparison to NFLlib [2], where they show clear improvements compared to NTL, indicates that an optimized implementation should provide a speedup by at least an order of magnitude.

7 Comparison

7.1 Comparison to DistDec (TCC'10)

We sketch an extension of the passively secure distributed decryption protocol Π_{DistDec} given by Bendlin and Damgård [11], which is used in SPDZ [20,22]. The main difference compared to our protocol is that this protocol requires zero-knowledge proofs to ensure correct computation at each step of the protocol to achieve active security instead of repeating the decryption procedure several times. The protocol works roughly as following:

1. Each party $\mathcal{D}_i$ samples uniform $E_{i,j}$ such that $\|E_{i,j}\|_\infty \leq 2^{40} B_{\mathtt{Dec}}/\xi p$ (for 40 bits statistical security) and computes the partial decryptions $t_{i,j} = s_i u_j + pE_{i,j}$ for each ciphertext $c_j = (u_j, v_j)$.
2. Each party $\mathcal{D}_i$ publish a zero-knowledge proof $\pi_{L_{i,j}}$ of the linear relation for $t_{i,j}$, using the lattice-based commitments together with their zero-knowledge proof of linear relations by Baum *et al.* [8].
3. Each party $\mathcal{D}_i$ use the amortized ZKP by Baum *et al.* [7] for batch-size N to prove that each $E_{i,j}$ is bounded by $2^{\mathtt{sec}} B_{\mathtt{Dec}}/\xi p$, given commitments $\boldsymbol{c}_{E_{i,j}}$.
4. The verifier checks the relations $(v_j - t_{0,j} - t_{1,j} \mod q) \equiv m_j \mod p$ and that all the zero-knowledge proofs are valid.

Elements t_j and commitments $\boldsymbol{c}_{E_{i,j}}$ are $N \log q$ and $2N \log q$ bits, respectively. Each proof of linearity $\pi_{L_{i,j}}$ is $6N \log(6\bar{\sigma})$ bits. The amortized proof is $540 \log(6\hat{\sigma})$ bits. The total size, for each $\mathcal{D}_i$, is

$$(3N \log q + 6N \log(6\bar{\sigma}) + 540 \log(6\hat{\sigma}))\tau \text{ bits.}$$

Then one party can split the key into $\xi = 2$ shares, run Π_{DistDec} on each key-share locally, and return the outputs from both $\mathcal{D}_1$ and $\mathcal{D}_2$ together with an additional proof that the key-splitting was correct. We based the estimate on the parameters from Table 1, with $\bar{\sigma} \approx 2^{16}$ and $\hat{\sigma} \approx 2^{66}$ (see e.g. Aranha *et al.* [5]

** https://github.com/tjesi/verifiable-decryption-in-the-head.

for details about proofs and sizes). However, the amortized proof is not exact, which means that we must increase q to $q \approx 2^{78}$ to ensure correct decryption. For security $\kappa = 128$ we also need to increase N to $N = 4096$. The proof is then of size $\approx 363\tau$ KB. We conclude that Π_{ZKPCD} is of equal size as Π_{DistDec} for $\lambda = 26$ and otherwise larger.

We do not have access to timings for this protocol. However, as the modulus is much larger, the dimension is twice the size, the zero-knowledge proofs include Gaussian sampling and rounds of aborts, we expect the protocol to be much slower than ours despite the large number of repetitions in our construction.

7.2 Comparison to Boschini *et al.* (PQ Crypto'20)

Boschini *et al.* [15] presents a zero-knowledge protocol for Ring-SIS and Ring-LWE. Their protocol can be used to prove knowledge of secrets or plaintexts, or prove correct decryption given a message and a BGV ciphertext. Concrete estimates for the latter are not given in the paper, but the number of constraints is higher for decryption than for the former. For a slightly smaller choice of parameters, a single proof of plaintext knowledge is of size 87 KB and takes roughly 3 minutes to compute. We conclude that the proof system by Boschini *et al.* will provide decryption proofs of equal size as protocol when $\lambda = 6$ and smaller otherwise. The time it takes to produce such a proof are several orders of magnitude slower than ours, making the system impossible to use in practice even for moderate sized sets of ciphertexts.

7.3 Comparison to Lyubashevsky *et al.* (PKC'21)

A recent publication by Lyubashevsky, Nguyen and Seiler [38] gives a verifiable decryption protocol for the Kyber encapsulation scheme [41]. Here, the encryption is over a rank 2 module over a ring of dimension $N = 256$ and modulus $q = 3329$ with secret and noise values bounded by $B_\infty = 2$. The proof of correct decryption of binary messages of dimension 256 is of size 43.6 KB, which of equal size as in our protocol for $\lambda = 3$. We note that the message space is smaller than in our protocol, mostly because we are forced to choose larger parameters to ensure correct decryption, and hence, we can not provide a proof of verifiable decryption for Kyber in particular. They do not provide timings, but we notice that the proof system use Gaussian sampling, rejection sampling, partially splitting rings and automorphisms – making the protocol very difficult to implement correctly and securely in practice.

7.4 Comparison to Silde (VOTING'22)

Silde [44] presents a direct verifiable decryption of BGV ciphertexts. The parameters are similar to our scheme, and the proof is of size 47 KB per ciphertext. This the same as in our scheme for $\lambda = 4$, ignoring the setup cost, while smaller for larger λ. The timing of the decryption protocol is 90 ms per ciphertext, which is equal to our timings for $\lambda = 23$ and otherwise up to 6 times faster for $\lambda = 128$.

Thanks

We thank Carsten Baum and the anonymous reviewers for helpful comments. Johannes Müller was supported by FNR under the CORE Junior project FP2 (C20/IS/14698166/FP2/Mueller).

References

1. Adida, B.: Helios: Web-based open-audit voting. In: van Oorschot, P.C. (ed.) USENIX Security 2008. pp. 335–348. USENIX Association (Jul / Aug 2008)
2. Aguilar Melchor, C., Barrier, J., Guelton, S., Guinet, A., Killijian, M.O., Lepoint, T.: NFLlib: NTT-based fast lattice library. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 341–356. Springer, Heidelberg (Feb / Mar 2016)
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
4. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2087–2104. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134104
5. Aranha, D.F., Baum, C., Gjøsteen, K., Silde, T.: Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. Cryptology ePrint Archive, Report 2022/422 (2022), https://ia.cr/2022/422
6. Attema, T., Lyubashevsky, V., Seiler, G.: Practical product proofs for lattice commitments. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 470–499. Springer, Heidelberg (Aug 2020)
7. Baum, C., Bootle, J., Cerulli, A., del Pino, R., Groth, J., Lyubashevsky, V.: Sublinear lattice-based zero-knowledge arguments for arithmetic circuits. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 669–699. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_23
8. Baum, C., Damgård, I., Lyubashevsky, V., Oechsner, S., Peikert, C.: More efficient commitments from structured lattice assumptions. In: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 11035, pp. 368–385. Springer, Heidelberg (Sep 2018)
9. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 495–526. Springer, Heidelberg (May 2020)
10. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2_4
11. Bendlin, R., Damgård, I.: Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 201–218. Springer, Heidelberg (Feb 2010)
12. Bertot, Y., Castéran, P., Huet, G., Paulin-Mohring, C.: Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions. Texts in theoretical computer science, Springer (2004)
13. Bettaieb, S., Schrek, J.: Improved lattice-based threshold ring signature scheme. In: Gaborit, P. (ed.) Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013. pp. 34–51. Springer, Heidelberg (Jun 2013)

14. Beullens, W.: Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 183–211. Springer, Heidelberg (May 2020)
15. Boschini, C., Camenisch, J., Ovsiankin, M., Spooner, N.: Efficient post-quantum SNARKs for RSIS and RLWE and their applications to privacy. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020. pp. 247–267. Springer, Heidelberg (2020)
16. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 3–33. Springer, Heidelberg (May 2019)
17. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012). https://doi.org/10.1145/2090236.2090262
18. Bruinderink, L.G., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 323–345. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53140-2_16
19. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO. Lecture Notes in Computer Science, vol. 740, pp. 89–105. Springer (1992)
20. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (Sep 2013)
21. Damgård, I., Orlandi, C., Takahashi, A., Tibouchi, M.: Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 99–130. Springer, Heidelberg (May 2021)
22. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012)
23. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg (Aug 1990)
24. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg (Aug 2016)
25. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1857–1874. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134028
26. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
27. Gordon, D.M.: A Survey of Fast Exponentiation Methods. J. Algorithms **27**(1), 129–146 (1998), https://doi.org/10.1006/jagm.1997.0913
28. Haines, T., Lewis, S.J., Pereira, O., Teague, V.: How not to prove your election outcome. In: 2020 IEEE Symposium on Security and Privacy. pp. 644–660. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00048
29. Haines, T., Müller, J.: SoK: Techniques for verifiable mix nets. In: Jia, L., Küsters, R. (eds.) CSF 2020 Computer Security Foundations Symposium. pp. 49–64. IEEE Computer Society Press (2020). https://doi.org/10.1109/CSF49147.2020.00012

30. Heiberg, S., Willemson, J.: Verifiable internet voting in Estonia. In: 6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014 (2014)
31. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press (Jun 2007). https://doi.org/10.1145/1250790.1250794
32. Kawachi, A., Tanaka, K., Xagawa, K.: Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 372–389. Springer, Heidelberg (Dec 2008)
33. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 158–189. Springer, Heidelberg (Apr / May 2018)
34. Ling, S., Nguyen, K., Stehlé, D., Wang, H.: Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 107–124. Springer, Heidelberg (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_8
35. Luo, F., Wang, K.: Verifiable decryption for fully homomorphic encryption. In: Chen, L., Manulis, M., Schneider, S. (eds.) ISC 2018. LNCS, vol. 11060, pp. 347–365. Springer, Heidelberg (Sep 2018)
36. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (Dec 2009)
37. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_43
38. Lyubashevsky, V., Nguyen, N.K., Seiler, G.: Shorter lattice-based zero-knowledge proofs via one-time commitments. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 215–241. Springer, Heidelberg (May 2021)
39. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_3
40. Peng, K., Boyd, C., Dawson, E.: Batch zero-knowledge proof and verification and its applications. ACM Trans. Inf. Syst. Secur. **10**(2), 6 (2007)
41. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
42. Shirazi, F., Simeonovski, M., Asghar, M.R., Backes, M., Diaz, C.: A survey on routing in anonymous communication protocols. ACM Comput. Surv. **51**(3) (Jun 2018). https://doi.org/10.1145/3182658, https://doi.org/10.1145/3182658
43. Shoup, V.: Ntl: A library for doing number theory (2021), https://libntl.org/index.html
44. Silde, T.: Verifiable Decryption for BGV. Workshop on Advances in Secure Electronic Voting (2022), https://ia.cr/2021/1693