





# Verifiable Decryption in the Head

Kristian Gjøsteen<sup>1</sup> , Thomas Haines<sup>1,2</sup>, Johannes Müller<sup>3</sup> ,  
Peter Rønne<sup>3,4</sup> , and Tjerand Silde<sup>1</sup> 

<sup>1</sup> Norwegian University of Science and Technology  
{kristian.gjosteen,tjerand.silde}@ntnu.no

<sup>2</sup> Australian National University  
thomas.haines@anu.edu.au

<sup>3</sup> University of Luxembourg  
johannes.mueller@uni.lu

<sup>4</sup> 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. To introduce our idea, we present a toy example for an ElGamal distributed decryption protocol that we also give a machine checked proof of, in addition to applying our method to lattices. 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 [SSA<sup>+</sup>18], decryption of ballots in electronic voting [HM20], and various uses of verifiable fully homomorphic encryption [LW18]. 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 [CP92] will do). Except for the recent publication by Lyubashevsky *et al.* [LNS21] (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 [Adi08] and the Estonian voting protocol [HW14] 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 [BD10, BS13, DPSZ12, DOT21, DHRW16, BKS19]. 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 [IKOS07] 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.

We prove in the interactive theorem prover Coq [\[BCHPM04\]](#) 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 [KTX08,LNSW13] 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 [ALS20,LNS21]. Correctly and securely implementing voting systems using primitives based on discrete logarithms is hard [HLPT20], 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 [BHLY16] or rejection sampling [EFGT17].

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 7.

## 1.2 Related Work

Verifiable decryption for ElGamal can be done by proving the equality of two discrete logarithms [CP92], and can be batched for significantly improved performance when decrypting many ciphertexts [Gor98,PBD07].

The "dual" Regev system [LPR13] 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 [DF90] and distributed decryption schemes are now well-understood, and many constructions exist [BD10], in particular those related to SPDZ [DKL<sup>+</sup>13,DPSZ12,KPR18]. 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 [BD10,BCOS20,LNS21,Sil22] in Section 8.

## 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  $\text{KeyGen}$ , encryption algorithm  $\text{Enc}$  and decryption algorithm  $\text{Dec}$ . We extend the notation with a predicate  $\text{KeyM}$  for key-matching which takes as input a public and secret key. We require for all matching public and secret keys  $\text{pk}, \text{sk}$  and all messages  $m$ , that  $\text{Dec}(\text{sk}, \text{Enc}(\text{pk}, m)) = m$  (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** ( $\text{Deal}$ ) takes as input a public key and corresponding secret key and outputs two *secret key shares* and some *auxiliary data*  $\text{aux}$ .

**The verify algorithm** ( $\text{Verify}$ ) 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** ( $\text{Play}$ ) takes as input a secret key share and a ciphertext and outputs a *decryption share*  $\text{ds}$ .

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

Intuitively, the protocol is *correct* if  $\text{Play}$  and  $\text{Rec}$  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  $(\text{pk}, \text{sk})$  s.t.  $\text{KeyM}(\text{pk}, \text{sk}) = 1$ , all  $c = \text{Enc}(\text{pk}, m)$ , any  $(\text{sk}_0, \text{sk}_1, \text{aux})$  output by  $\text{Deal}(\text{pk}, \text{sk})$ , then, for  $i = 0, 1$ ,  $\text{Verify}(\text{pk}, \text{aux}, i, \text{sk}_i) = 1$ , and*

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

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  $\text{Verify}$  algorithm), then the  $\text{Play}$  and  $\text{Rec}$  will collectively recover the encrypted message.

**Definition 2 (Integrity).** *A distributed decryption protocol has integrity if there exists an efficient algorithm (named  $\text{FindKey}$  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  $\text{pk}$ , ciphertexts  $c = \text{Enc}(\text{pk}, m)$ , secret key shares  $(\text{sk}_1, \text{sk}_2)$ , and auxiliary data  $\text{aux}$  and  $\text{sk}$  output by  $\text{FindKey}(\text{pk}, \text{sk}_0, \text{sk}_1, \text{aux})$  satisfying  $\text{Verify}(\text{pk}, \text{aux}, i, \text{sk}_i) = 1$ , for  $i = 0, 1$ , we have that*

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

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.

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

**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  $\mathbf{c} = (c_0, \dots, c_\tau)$ ,  $\mathbf{m} = (m_0, \dots, m_\tau)$  such that  $\{m_j = \text{Dec}(\mathbf{sk}, c_j)\}_{j=1}^\tau$ . The simulatability advantage of  $\mathcal{A}$  is

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

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

## 2.1 Toy Example: Distributed ElGamal

We briefly recall ElGamal encryption for a given cyclic group  $\mathbb{G}$  of prime order  $p$  with generator  $g$  and give a toy decryption example.

**Key generation** (`KeyGen`) samples  $x$  from  $\mathbb{Z}_p^*$  and return  $(g^x, x)$ .

**Encryption** (`Enc`) takes as input a public key  $\mathbf{pk} \in \mathbb{G}$  and message  $m \in \mathbb{G}$ , samples  $r$  from  $\mathbb{Z}_p^*$ , and returns  $(g^r, \mathbf{pk}^r m)$ .

**Decryption** (`Dec`) takes as input a secret key  $x \in \mathbb{Z}_p^*$  and ciphertext  $(c_1, c_2)$ , and returns  $c_2/c_1^x$ .

**Keymatch** (`KeyM`) takes as input a public key  $\mathbf{pk} \in \mathbb{G}$  and a secret key  $x \in \mathbb{Z}_p^*$  and returns 1 if  $g^x = \mathbf{pk}$  and otherwise 0.

We will now give a distributed decryption protocol for ElGamal. This uses a  $(2, 2)$ -secret sharing of the decryption key, and it works because of ElGamal's key-homomorphic property.

**The dealer algorithm** (*Deal*) takes as input a public key  $\mathbf{g}^x$  and corresponding secret key  $x$ , samples  $x_0$  from  $\mathbb{Z}_p^*$ , sets  $x_1 = x - x_0$  and returns  $(x_0, x_1, \mathbf{aux} = (\mathbf{g}^{x_0}, \mathbf{g}^{x_1}))$ .

**The verify algorithm** (*Verify*) takes as input a public key  $\mathbf{pk}$ , auxiliary data  $\mathbf{aux} = (\mathbf{aux}_0, \mathbf{aux}_1)$ , an index  $i$  and a secret key share  $x_i$  and outputs 1 iff  $(\mathbf{g}^{x_i} = \mathbf{aux}_i) \wedge (\mathbf{pk} = \mathbf{aux}_0 \mathbf{aux}_1)$ .

**The player algorithm** (*Play*) takes as input a secret key share  $x_i$  and a ciphertext  $(c_1, c_2)$  and outputs *decryption share*  $c_1^{x_i}$ .

**The reconstruction algorithm** (*Rec*) takes as input a ciphertext  $(c_1, c_2)$  and decryption shares  $(t_0, t_1)$  and outputs  $c_2/(t_0 t_1)$ .

**Correctness.** Substituting ElGamal into the definition of correctness, for  $(g^x, x)$  and  $(x_0, x_1, (g^{x_0}, g^{x_1})) \leftarrow \text{Deal}(g^x, x)$ , we get that the verify algorithm accepts both secret key shares and for any ciphertext  $(g^r, g^{x^r} m)$ , we get that

$$((g^x)^r m) / ((g^r)^{x_0} (g^r)^{x_1}) = (g^r)^x m (g^r)^{-x_0 - x_1} = m,$$

so correctness holds unconditionally.

**Integrity.** *FindKey* takes as input two key shares  $x_0, x_1$  and outputs  $x = x_0 + x_1$ . Again, if the verify algorithm accepts both secret key shares, then we know that  $g^x = g^{x_0} g^{x_1}$  and unconditional integrity follows from the computations above.

**Privacy.** Simulators *DealSim* and *PlaySim* work as follows:

- *DealSim* takes the public key  $\mathbf{pk}$  and a bit  $i$ , samples  $x_i$  from  $\mathbb{Z}_p^*$  and returns (wlog.)  $(x_i, (\mathbf{g}^{x_i}, \mathbf{pk}/\mathbf{g}^{x_i}))$ . It is clear that the auxiliary data and secret key from the simulator have the same distribution as the *Deal*.
- *PlaySim* takes as input public key  $\mathbf{pk}$ , secret key  $x_i$ , ciphertext  $(c_1, c_2)$ , and message  $m$  and returns a decryption share  $c_2/(c_1^{x_i} m)$ . Since  $m$  is the message encrypted in the ciphertext this is a perfect simulation if  $m$  is the correct decryption.

Note that these simulators are perfect due to ElGamal's elegant homomorphic structure, both with respect to keys and messages.

### 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  $\lambda$  sharings of the secret key by calling the Deal algorithm  $\lambda$  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  $\lambda$ . 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.

*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.

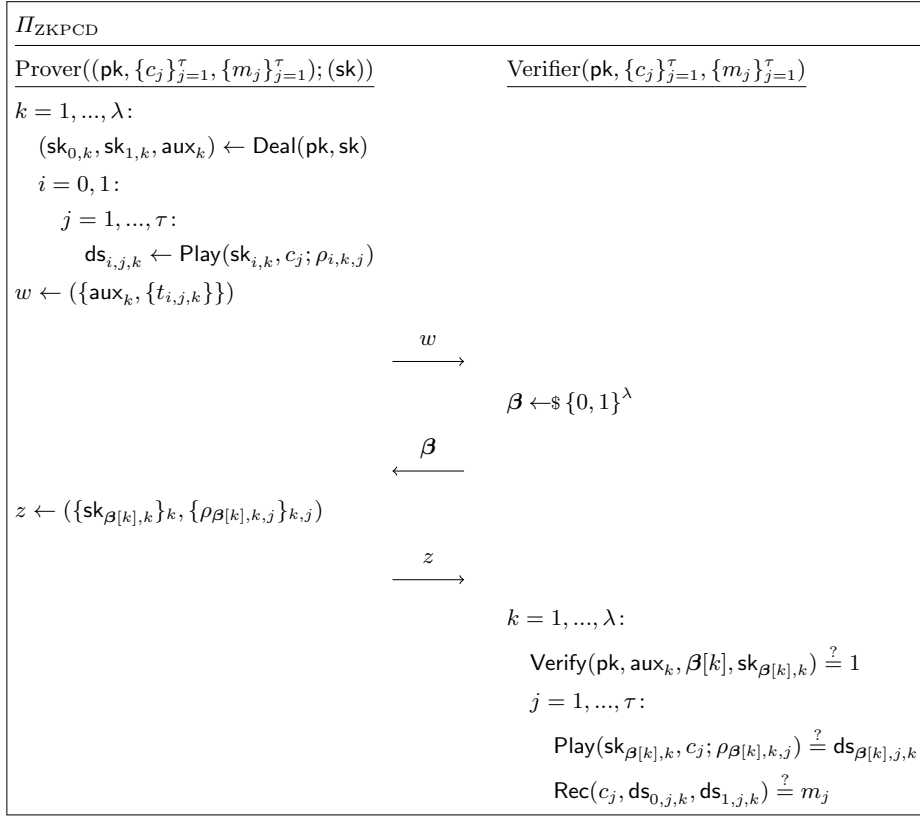
*Special Soundness.* By rewinding, any cheating prover with a significant success probability can be used to create two accepting conversations  $(w, \beta, z)$  and  $(w, \beta', z')$ , with  $\beta \neq \beta'$ . From this it follows that  $\beta[k] \neq \beta'[k]$  for at least one  $k$ , and the 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 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  $(\text{pk}, \{c_j\}_{j=1}^\tau, \{m_j\}_{j=1}^\tau)$  and the challenge  $\beta$ : First, for  $i = 1, \dots, \lambda$ , we let  $(\text{aux}_i, \text{sk}_{\beta[i],i}) \leftarrow \text{DealSim}(\text{pk}, \beta[i])$  and, for  $j = 1, \dots, \tau$ , we let  $\text{ds}_{\beta[i],j,i} \leftarrow \text{PlaySim}(\text{pk}, \text{sk}_{\beta[i],i}, c_j, m_j)$  and  $\text{ds}_{1-\beta[i],j,i} \leftarrow \text{Play}(\text{pk}, \text{sk}_{\beta[i],i}, c_j)$ . The proof transcripts is then  $((\text{pk}, \{c_j\}_{j=1}^\tau, \{m_j\}_{j=1}^\tau), (\text{aux}_i, \text{ds}_{0,j,i}, \text{ds}_{1,j,i}), \beta, \text{sk}_{\beta[i],i})$ . This is computationally indistinguishable from the honest transcripts if the distributed decryption protocol is simulatable.

## 4 Machine Checked Proofs

We adopt the definition of a sigma protocol from [HGT19] but do not require that the simulator always produces accepting transcripts when the statement is





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

not in the language. This does not affect on our intended use cases but prevents us from applying the standard transform to a disjunctive proof.

- We formally define an encryption scheme along the lines above in the paper but with perfect correctness. This can be found in the attached source in the Module Type EncryptionScheme.
- We formally define a distributed decryption schemes as a functor on encryptions schemes as above in the paper. However, we require perfect correctness, integrity and simulatability. (Module Type DistributedDecryption)
- We describe the transform for an arbitrary distributed decryption scheme and prove that the result is a sigma protocol for correct decryption. (Module ProofOfDecryption)
- We define the ElGamal cryptosystem and distributed decryption protocol and prove they satisfy the respective definitions. (ElGamal, DDElGamal).

The source code written in Coq is available online<sup>†</sup>.

We are unable to do better because no interactive theorem provides good support for cryptographic arguments and support for lattice primitives. Nevertheless, our work is an important step in the direction of proving the full result if and when interactive theorem provers are ready.

## 5 Background: Lattice-Based Cryptography

### 5.1 The Cyclotomic Ring $R_q$

Let  $N$  be a power of 2 and  $q$  a prime such that  $q \equiv 1 \pmod{2N}$ . Then we define the rings  $R = \mathbb{Z}[X]/\langle X^N + 1 \rangle$  and  $R_q = R/qR$ , that is,  $R_q$  is the ring of polynomials modulo  $X^N + 1$  with integer coefficients modulo  $q$ . This way,  $X^N + 1$  splits completely into  $N$  irreducible factors modulo  $q$ , which allows for very efficient computation in  $R_q$  due to the number theoretic transform (NTT) [LN16].

We define the norms of elements

$$f(X) = \sum \alpha_i X^i \in R$$

to be the norms of the coefficient vector as a vector in  $\mathbb{Z}^N$ :

$$\|f\|_1 = \sum |\alpha_i|, \quad \|f\|_2 = \left( \sum \alpha_i^2 \right)^{1/2}, \quad \|f\|_\infty = \max\{|\alpha_i|\}.$$

For an element  $\bar{f} \in R_q$  we choose coefficients as the representatives in  $[-\frac{q-1}{2}, \frac{q-1}{2}]$ , and then compute the norms as if  $\bar{f}$  is an element in  $R$ . For vectors  $\mathbf{a} = (a_1, \dots, a_k) \in R^k$  we define the norms to be

$$\|\mathbf{a}\|_1 = \sum \|a_i\|_1, \quad \|\mathbf{a}\|_2 = \left( \sum \|a_i\|_2^2 \right)^{1/2}, \quad \|\mathbf{a}\|_\infty = \max\{\|a_i\|_\infty\}.$$

### 5.2 Knapsack Problems

We first define the Search Knapsack problem in the  $\ell_2$  norm, also denoted as  $\text{SKS}^2$ . The  $\text{SKS}^2$  problem is exactly the Ring-SIS problem in its Hermite Normal Form.

**Definition 4.** *The  $\text{SKS}_{N,q,\beta}^2$  problem is to find a short vector  $\mathbf{x}$  of  $\ell_2$  norm less than or equal to  $\beta$  in  $R_q^2$  satisfying  $[a \ 1] \cdot \mathbf{x} = 0$  for a given uniformly random  $a$  in  $R_q$ . An algorithm  $\mathcal{A}$  has advantage  $\epsilon$  in solving the  $\text{SKS}_{N,q,\beta}^2$  problem if*

$$\Pr \left[ \begin{array}{l} [a \ 1] \cdot \mathbf{x} = 0 \\ \wedge \ \|x_i\|_2 \leq \beta \mid \mathbf{0} \neq \mathbf{x} \in R_q^2 \leftarrow \mathcal{A}(a) \end{array} \right] \geq \epsilon.$$

Additionally, we define the Decisional Knapsack problem in the  $\ell_\infty$  norm denoted as  $\text{DKS}^\infty$ . The  $\text{DKS}^\infty$  problem is equivalent to the Ring-LWE problem when the number of samples is limited.

<sup>†</sup> [www.dropbox.com/s/mn9gfmw3utkffyq](http://www.dropbox.com/s/mn9gfmw3utkffyq)

**Definition 5.** The  $\text{DKS}_{N,q,\beta}^\infty$  problem is to distinguish the distribution  $[a \ 1] \cdot \mathbf{x}$ , for a short  $\mathbf{x}$ , from the uniform distribution when given uniformly random  $a$  in  $R_q$ . An algorithm  $\mathcal{A}$  has advantage  $\epsilon$  in solving the  $\text{DKS}_{N,q,\beta}^\infty$  problem if

$$\begin{aligned} & |\Pr[b = 1 \mid a \leftarrow R_q; \mathbf{x} \leftarrow R_q^2 \text{ s.t. } \|\mathbf{x}\|_\infty \leq \beta; b \leftarrow \mathcal{A}(a, [a \ 1] \cdot \mathbf{x})] \\ & - \Pr[b = 1 \mid a \leftarrow R_q; u \leftarrow R_q; b \leftarrow \mathcal{A}(a, u)]| \geq \epsilon. \end{aligned}$$

See [LM06, LPR10] for more details about knapsack problems.

### 5.3 BGV Encryption

We present a plain version of the BGV encryption scheme by Brakerski, Gentry and Vaikuntanathan [BGV12]. 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  $\kappa$  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  $\text{pk} = (a, b) = (a, as + pe)$  and the secret key  $\text{sk} = (s, e)$ .
- **Enc**, on input the public key  $\text{pk} = (a, b)$  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  $\text{sk} = (s, e)$  and a ciphertext  $c = (u, v)$ , outputs the message  $m = (v - su \bmod q) \bmod p$  in  $R_p$ .

The decryption algorithm is correct as long as the norm  $\max\|v - su\|_\infty = B_{\text{dec}} < [q/2]$ . It follows that the BGV encryption scheme is secure against chosen plaintext attacks if the  $\text{DKS}_{N,q,\beta}^\infty$  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 [BD10] used in the MPC-protocols by Damgård *et al.* [DKL<sup>+</sup>13, DPSZ12]. When decrypting, we assume that each decryption server  $\mathcal{D}_j$ , for  $1 \leq j \leq \xi$ , has a uniformly random share  $\text{sk}_j = s_j$  of the secret key  $\text{sk} = (s, e)$  such that  $s = s_1 + s_2 + \dots + s_\xi$ . Then they partially decrypt in the following way:

- **DistDec**, on input a secret key-share  $\text{sk}_j = s_j$  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  $\|E_j\|_\infty \leq 2^{\text{sec}}(B_{\text{dec}}/p\xi)$  for some statistical security parameter  $\text{sec}$  and upper error-bound  $\max\|v - su\|_\infty \leq B_{\text{dec}}$ , then outputs  $\text{ds}_j = t_j = m_j + pE_j$ .

We obtain the full decryption of the ciphertext  $(u, v)$  as  $m \equiv (v - t \bmod q) \bmod p$ , where  $t = t_1 + t_2 + \dots + t_\xi$ . This will give the correct decryption as long as the noise  $\max\|v - t\|_\infty \leq (1 + 2^{\text{sec}})B_{\text{dec}} < [q/2]$  (see [DKL<sup>+</sup>13, Appendix G]). Here,  $t$  will be indistinguishable from random except with probability  $2^{-\text{sec}}$ .

## 5.4 Lattice-Based Commitments

We first define a commitment scheme and its security.

**Definition 6 (Commitment Scheme).** *A commitment scheme consists of three algorithms: key generation ( $\text{KeyGen}$ ), commitment ( $\text{Com}$ ) and opening ( $\text{Open}$ ), where*

- $\text{KeyGen}$ , on input security parameter  $1^\lambda$ , outputs public parameters  $\text{pp}$ ,
- $\text{Com}$ , on input message  $m$ , outputs commitment  $c$  and opening  $r$ ,
- $\text{Open}$ , on input  $m$ ,  $c$  and  $r$ , outputs either 0 or 1,

and the public parameters  $\text{pp}$  are implicit inputs to  $\text{Com}$  and  $\text{Open}$ .

**Definition 7 (Completeness).** *We say that the commitment scheme is complete if an honestly generated commitment is accepted by the opening algorithm. Hence, we want that*

$$\Pr \left[ \text{Open}(m, c, r) = 1 : \begin{array}{l} \text{pp} \leftarrow \text{KeyGen}(1^\lambda) \\ (c, r) \leftarrow \text{Com}(m) \end{array} \right] = 1,$$

where the probability is taken over the random coins of  $\text{KeyGen}$  and  $\text{Com}$ .

**Definition 8 (Hiding).** *We say that a commitment scheme is hiding if an adversary  $\mathcal{A}$ , after choosing two messages  $m_0$  and  $m_1$  and receiving a commitment  $c$  to either  $m_0$  or  $m_1$  (chosen at random), cannot distinguish which message  $c$  is a commitment to. Hence, we want that*

$$\left| \Pr \left[ b = b' : \begin{array}{l} \text{pp} \leftarrow \text{KeyGen}(1^\lambda) \\ (m_0, m_1, \text{st}) \leftarrow \mathcal{A}(\text{pp}) \\ b \xleftarrow{\$} \{0, 1\}, c \leftarrow \text{Com}(m_b) \\ b' \leftarrow \mathcal{A}(c, \text{st}) \end{array} \right] - \frac{1}{2} \right| \leq \text{negl},$$

where the probability is taken over the random coins of  $\text{KeyGen}$  and  $\text{Com}$ .

**Definition 9 (Binding).** *We say that a commitment scheme is binding if an adversary  $\mathcal{A}$ , after creating a commitment  $c$ , cannot find two valid openings to  $c$  for different messages  $m$  and  $\hat{m}$ . Hence, we want that*

$$\Pr \left[ \begin{array}{l} m \neq \hat{m} \\ \text{Open}(m, c, r) = 1 \\ \text{Open}(\hat{m}, c, \hat{r}) = 1 \end{array} : \begin{array}{l} \text{pp} \leftarrow \text{KeyGen}(1^\lambda) \\ (c, m, r, \hat{m}, \hat{r}) \leftarrow \mathcal{A}(\text{pp}) \end{array} \right] \leq \text{negl},$$

where the probability is taken over the random coins of  $\text{KeyGen}$ .

We note that the public key in the BGV encryption scheme is essentially a commitment to the secret key. In general, ignoring the constant  $p$ , the value  $b = as + e$  is a commitment to a short random secret  $s$  with randomness  $e$  if  $e$  is short and  $a$  is a uniformly random public element.

More formally, let  $q$  be a prime, let  $R_q$  be defined as above for a fixed  $N$  and let  $B_\infty \in \mathbb{N}$  be a bound. These are the public parameters  $\text{pp}$ . The commitment scheme consists of three algorithms: key generation ( $\text{KeyGen}$ ), commit ( $\text{Com}$ ) and open ( $\text{Open}$ ), where

- **KeyGen** samples an element  $a' \leftarrow_{\$} R_q$  uniformly at random and outputs the public commitment key  $\text{pk}' = a'$ .
- **Com**, on input the public key  $\text{pk}' = a'$  and a pseudo-random message  $m$  in  $R_q$  such that  $\|m\|_{\infty} \leq B_{\infty}$ , samples a short  $r_m \leftarrow_{\$} R_q$  such that  $\|r_m\|_{\infty} \leq B_{\infty}$ , outputs commitment  $c_m = a'm + r_m$  and opening  $d_m = (m, r_m)$ .
- **Open**, on input a commitment  $c_m$  and an opening  $d_m = (m, r_m)$ , checks if  $\max(\|m\|_{\infty}, \|r_m\|_{\infty}) \leq B_{\infty}$  and  $c_m = a'm + r_m$ , and outputs 1 if both checks hold and otherwise 0.

It follows directly that the commitment scheme is (computationally) hiding if  $\text{DKS}_{N,q,B_{\infty}}^{\infty}$  is hard and (computationally) binding if  $\text{SKS}_{N,q,\sqrt{2N} \cdot B_{\infty}}^2$  is hard.

### 5.5 Zero-Knowledge Proof of Shortness

We present the Stern-based [Ste94] zero-knowledge proof of knowledge protocol by Kawachi *et al.* [KTX08] and Ling *et al.* [LNSW13]. We will later use this to prove that we know a valid opening  $d_m$  of a commitment  $c_m$  without leaking any information about the message nor the randomness. We denote the protocol by  $\Pi_{\text{ZKPoS}}$ .

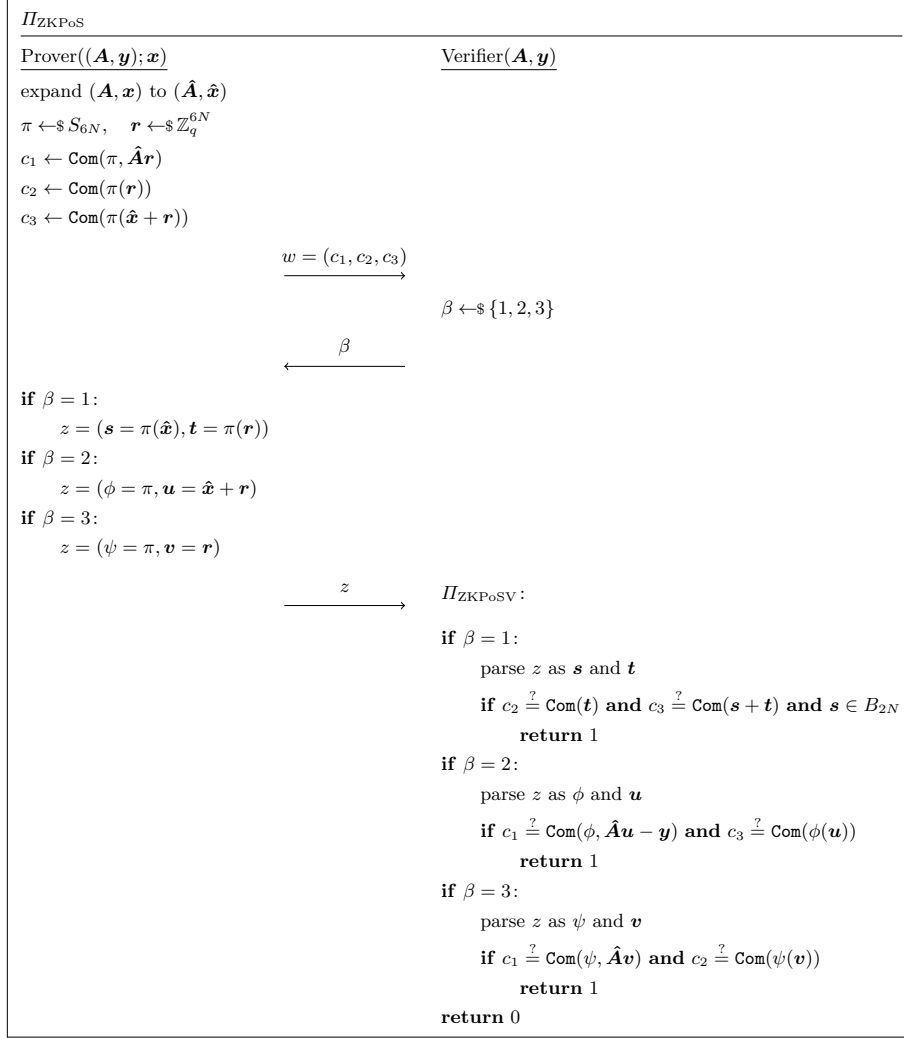
Note that multiplication with a polynomial  $a'$  in  $R_q$  can be re-written as a matrix-vector product by a negacyclic  $N \times N$ -matrix  $\mathbf{A}'$  over  $\mathbb{Z}_q$ . Define  $\mathbf{A} = [\mathbf{A}' \quad \mathbf{I}_N]$ , and let  $B_{\infty} = 1$  for simplicity (it can be generalized to any  $B_{\infty}$ , see [LNSW13]). We give a zero-knowledge protocol for the following relation:

$$R_{\text{DKS}_{N,q,1}^{\infty}} = \{((\mathbf{A}, \mathbf{y}); \mathbf{x}) : \mathbf{A}\mathbf{x} = \mathbf{y} \pmod{q} \wedge \|\mathbf{x}\|_{\infty} = 1\}.$$

Furthermore, let  $B_m$  be the set of all vectors  $\hat{\mathbf{x}}$  of length  $3m$  with  $m$  coordinates of each element in  $\{0, 1, -1\}$ . It is then trivial to extend any witness  $\mathbf{x}$  of the relation  $R_{\text{DKS}_{N,q,1}^{\infty}}$  to a vector  $\hat{\mathbf{x}}$  in  $B_{2N}$  by appending values and extending the matrix  $\mathbf{A}$  to the matrix  $\hat{\mathbf{A}} = [\mathbf{A} \quad \mathbf{0}_{N \times 4N}]$ . It follows that  $\mathbf{A}\mathbf{x} = \mathbf{y} \pmod{q} \wedge \|\mathbf{x}\|_{\infty} = 1$  if and only if  $\hat{\mathbf{A}}\hat{\mathbf{x}} = \mathbf{y} \pmod{q} \wedge \hat{\mathbf{x}} \in B_{2N}$ . Let  $S_{6N}$  be the set of permutations on  $6N$  symbols. The full protocol is given in Figure 3.

This protocol has soundness  $2/3$ , and hence, must be repeated  $\mu = \lambda \cdot \ln(2)/\ln(3/2)$  times to achieve soundness  $2^{-\lambda}$ . However, the protocol is very simple and lightweight in computation. We observe that the commitments  $c_1, c_2, c_3$  are commitments to random values, and the commitments does not need any structure. Hence, we can compute the commitments as plain hashes of the committed values. Furthermore, we only need to sample a uniformly random permutation  $\pi$  from  $S_{6N}$  and a uniformly random vector  $\mathbf{r}$  from  $\mathbb{Z}_q^{6N}$ . Compared to other lattice-based zero-knowledge protocols, there are no Gaussian sampling, no rejection sampling, no use of partially splitting rings or automorphisms.

We restrict permutations  $\pi$  to lie in the subgroup  $H$  generated by sign swaps and the transpositions  $\{(i \ i + 6N) \mid i \text{ from } 1 \text{ to } 6N\}$ . This subgroup has  $8^{6N}$  elements which can be represented by  $18N$  bits. Each vector in  $\mathbb{Z}_q^{6N}$  can be represented by  $6N \log q$  bits, each vector in  $B_{2N}$  can be represented by  $12N$  bits, and each commitment can be represented by  $2\kappa$  bits. As  $\beta$  is uniformly



**Fig. 3.** Zero-knowledge proof of shortness.

distributed we estimate the proof size assuming that each response will appear a third of the times each. The total proof, denoted  $\pi_S$ , is of size

$$|\pi_S| = (6\kappa + 16N + 6N \log q)\mu \text{ bits.} \quad (1)$$

We finally note that the protocol can be improved using the combinatorial extensions by Beullens [Beu20], reducing the size of the proof by a factor 10 without much extra computational work nor increased complexity in the implementation.

## 6 Zero-Knowledge Protocol of Correct Decryption

### 6.1 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  $\kappa$ , soundness parameter  $\lambda$ , bound  $B_\infty$  and maximal ciphertext error-bound  $B_{\text{Dec}}$ . 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  $(\Pi_{\text{ZKPoS}}, \Pi_{\text{ZKPoSV}})$  be a non-interactive zero-knowledge protocol for the following relation:

$$R_{\text{DKS}}^{\infty, N, q, 1} = \{((\mathbf{A}, \mathbf{y}); \mathbf{x}) : \mathbf{A}\mathbf{x} = \mathbf{y} \pmod{q} \wedge \|\mathbf{x}\|_\infty = 1\}.$$

*Scheme.* We present a distributed decryption version of the BGV encryption scheme [BGV12], where **KeyGen**, **Enc** and **Dec** are defined in Section 5.3.

**The dealer algorithm** (**Deal**) takes as input a public key  $\mathbf{pk} = (a, b)$  and corresponding secret key  $\mathbf{sk} = (s, e)$ , 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  $c_{s_i} = \text{Com}(s_i)$ ,  $c_{e_i} = \text{Com}(e_i)$ , and computes  $b_i = as_i + pe_i$  so that  $b = b_0 + b_1$ . Finally, it computes non-interactive zero-knowledge proofs  $\pi_{S_i}$  proving that the sums  $s_0 + s_1$  and  $e_0 + e_1$  are short (see details in Section 7). It outputs key shares  $sk_0 = (s_0, e_0)$ ,  $sk_1 = (s_1, e_1)$  and  $\mathbf{aux} = (b_0, b_1, c_{s_0}, c_{s_1}, c_{e_0}, c_{e_1}, \pi_{S_0}, \pi_{S_1})$ .

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

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

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

## 6.2 Security

**Theorem 1 (Correctness).** *The distributed decryption scheme in 6.1 is correct with respect to Definition 1 when  $\max\|v - t\|_\infty \leq (1 + 2^{\text{sec}})B_{\text{Dec}} < \lfloor q/2 \rfloor$ .*

**Theorem 2 (Integrity).** *Suppose the protocol  $\Pi_{\text{ZKPoS}}$  is (computationally) sound and that  $\text{Com}$  is (computationally) binding. Let  $\mathcal{A}_0$  be an adversary against integrity of the distributed decryption scheme with advantage  $\epsilon_0$ , and let  $\lambda$  be the number of rounds in the protocol. Then there exists adversaries  $\mathcal{A}_1$  and  $\mathcal{A}_2$  against soundness of  $\Pi_{\text{ZKPoS}}$  and binding of  $\text{Com}$ , respectively, with advantages  $\epsilon_1$  and  $\epsilon_2$ , 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  $\text{Verify}$  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  $\pi_S$  of the sums  $s_0 + s_1$  and  $e_0 + e_1$  are accepted. If either of the key shares are incorrect then  $\text{Verify}$  accept with probability 0, and if the key shares are correct, then  $\text{Rec}$  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  $\text{Verify}$  then reduces to the soundness of the zero-knowledge protocol, and an attacker  $\mathcal{A}_0$  against this part of the protocol with advantage  $\epsilon_0$  can be turned into an attacker  $\mathcal{A}_1$  against  $\Pi_{\text{ZKPoS}}$  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,  $\text{Verify}$  only accepts if both the opening of the commitments are correct and the zero-knowledge proof of shortness verifies. Hence, an attacker  $\mathcal{A}_0$  that is able to produce valid openings and proofs with advantage  $\epsilon_0$  can be turned into an attacker  $\mathcal{A}_2$  against  $\text{Com}$  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.  $\square$

**Theorem 3 (Privacy).** *Suppose the protocol  $\Pi_{\text{ZKPoS}}$  is (statistically) honest-verifier zero-knowledge, that  $\text{Com}$  is (computationally) hiding and that  $\text{Enc}$  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  $\epsilon_0$  there exists an adversary  $\mathcal{A}_2$  against hiding for the commitment scheme with advantage  $\epsilon_2$ , an adversary  $\mathcal{A}_3$  against CPA security for the encryption scheme with advantage  $\epsilon_3$ , and a distinguisher  $\mathcal{A}_1$  for the simulator of  $\Pi_{\text{ZKPoS}}$  with advantage  $\epsilon_1$ , 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  $\text{Sim}_{\text{Short}}$  be a simulator for  $\Pi_{\text{ZKPoS}}$ . We present a simulator  $\text{DealSim}$  for the Deal-algorithm and a simulator  $\text{PlaySim}$  for the Play-algorithm in Figure 4.

$\text{DealSim}(\text{pk} = (a, b), i)$	$\text{PlaySim}(\text{sk}_{1-i} = (s_{1-i}, e_{1-i}), c = (u, v), i, m)$
$i = 0, 1: s_i^* \leftarrow_{\$} R_q, \quad e_i^* \leftarrow_{\$} R_q$ $b_i^* = as_i^* + pe_i^*, \quad b_{1-i}^* = b - b_i^*$ $c_{s_i}^* \leftarrow \text{Com}(s_i^*), c_{s_{1-i}}^* \leftarrow \text{Com}(s_{1-i})$ $c_{e_i}^* \leftarrow \text{Com}(e_i^*), c_{e_{1-i}}^* \leftarrow \text{Com}(e_{1-i})$ $\pi_S^* \leftarrow \text{Sim}_{\text{Short}}(c_{s_i}^*, c_{s_{1-i}}^*, c_{e_i}^*, c_{e_{1-i}}^*)$ $\text{aux}^* \leftarrow (b_0^*, b_1^*, c_{s_0}^*, c_{s_1}^*, c_{e_0}^*, c_{e_1}^*, \pi_S^*)$ <b>return</b> $(\text{sk}_i^* = (s_i^*, e_i^*), \text{aux}^*)$	$E_{1-i} \leftarrow_{\$} \mathbb{E}$ $t_{1-i} = s_{1-i}u + pE_{1-i}$ $t_i^* = v - m - t_{1-i} \pmod p$ <b>return</b> $(\text{ds}_i^* = t_i^*)$

**Fig. 4.** Simulators  $\text{DealSim}$  and  $\text{PlaySim}$ .

**DealSim:** We create the simulator in three steps. We first replace  $\pi_S$  by the simulated proof  $\pi_S^*$  produced by  $\text{Sim}_{\text{Short}}$ . An attacker  $\mathcal{A}_0$  with advantage  $\epsilon_0$  against this change can be turned into an attacker  $\mathcal{A}_1$  against the simulator  $\text{Sim}_{\text{Short}}$  of protocol  $\Pi_{\text{ZKPoS}}$  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  $\text{Verify}$  outputs 1. An attacker  $\mathcal{A}_0$  with advantage  $\epsilon_0$  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  $\epsilon_0$  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  $(v - t_0 - t_1 \pmod q) \pmod p = m$ . This ensures that  $\text{Rec}$  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  $\kappa$ .  $\square$

### 6.3 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 5. 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  $\mathbf{pk} = (a, b = as + pe)$ , where  $\max(\|s\|_\infty, \|e\|_\infty) \leq B_\infty$ . The secret key  $\mathbf{sk} = (s, e)$  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  $\lambda$ , the prover does the following for  $k = 1, \dots, \lambda$ . First, it runs the Deal algorithm on  $\mathbf{sk}$  and  $\mathbf{pk}$  to produce  $\mathbf{sk}_{0,k}, \mathbf{sk}_{1,k}$  and  $\mathbf{aux}_k$ . It uses  $\Pi_{\text{ZKPoS}}$  to prove that the shares are correctly computed. Then, for  $i = 0, 1$  and each  $j = 1, \dots, \tau$ , it runs the Play algorithm on each key-share  $\mathbf{sk}_{i,k}$  and ciphertext  $c_j$  to produce  $t_{0,j,k}$  and  $t_{1,j,k}$ . Finally, it sends  $w \leftarrow (\{\mathbf{aux}_k, \{t_{i,j,k}\}_{i=0, j=1}^{1,\tau}\}_{k=1}^\lambda)$  to end the commitment phase.

*Challenge phase.* The verifier independently samples a random binary challenge vector  $\beta$  of length  $\lambda$ . It sends  $\beta$  to the prover.

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

*Verification phase.* For each  $k = 1, \dots, \lambda$ , the verifier runs the Verify algorithm to make sure that the openings of  $s_{\beta[k],k}$  and  $e_{\beta[k],k}$  are valid, check that all shares of the public key are computed correctly as  $b_{\beta[k],k} = as_{\beta[k],k} + pe_{\beta[k],k}$ , verify the public key  $b = b_{0,k} + b_{1,k}$  and ensure that each  $\pi_{S_{i,k}}$  is valid. Further, for each  $j = 1, \dots, \tau$ , 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 [FS87] 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  $\lambda$  initially can be very small in the interactive case, while it should be (approximately) as large as the security parameter  $\kappa$  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  $\lambda_I$  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.

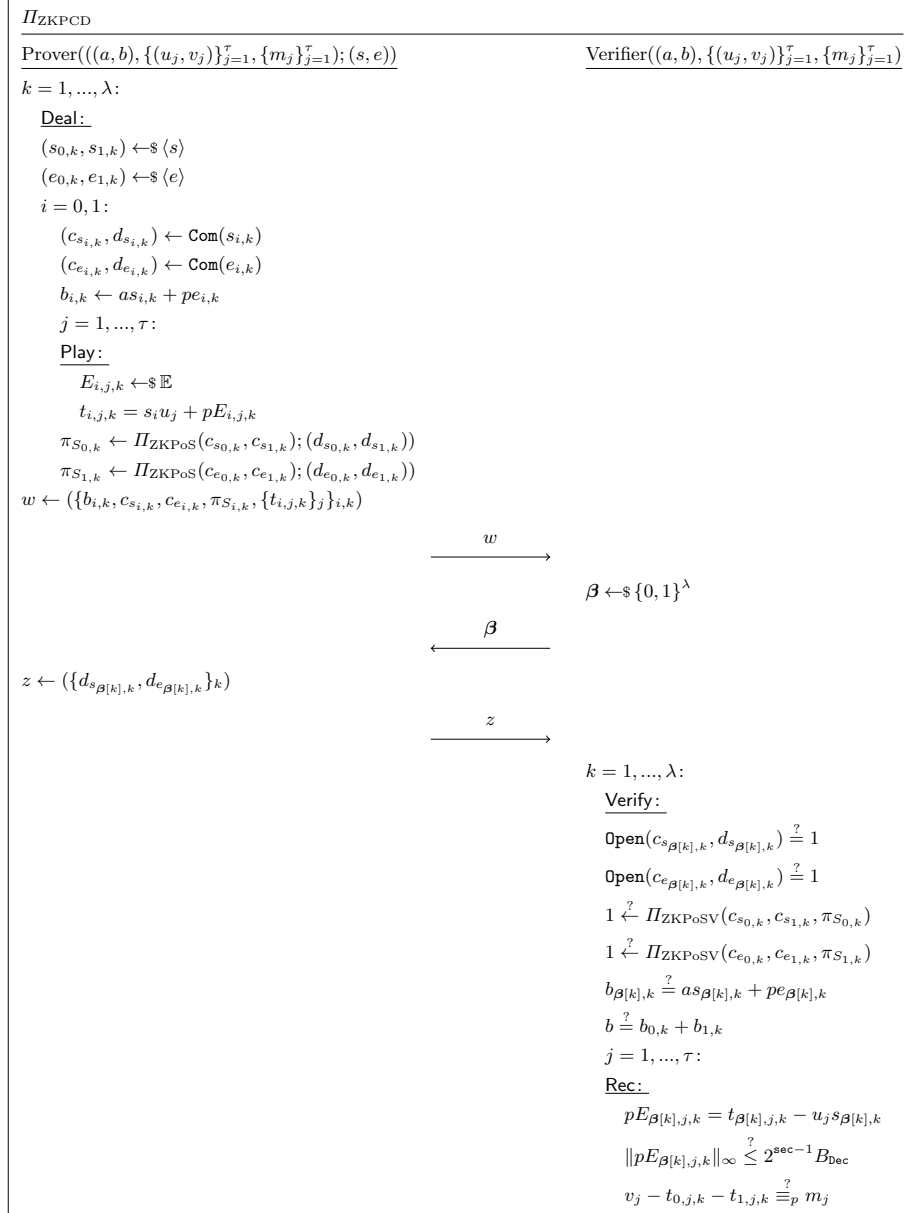


Fig. 5. Zero-knowledge proof of correct decryption.

## 7 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 Figure 5. In particular, we use a commitment in the first message, and then send only the values that the verifier cannot recompute himself in the 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.

### 7.1 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  $B_\infty$  is of size  $N \log B_\infty$  bits. We let  $H$  be a collision resistant hash-function with output of length  $2\kappa$ . Note that the soundness parameter  $\lambda$  may be chosen independently of, and in particular smaller than, the security parameter  $\kappa$ .

*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  $w = H(\{b_{0,k}, b_{1,k}, c_{s_{0,k}}, c_{s_{1,k}}, c_{e_{0,k}}, c_{e_{1,k}}, \{t_{i,j,k}\}_{i=0,j=1}^{1,\tau}\}_{k=1}^\lambda)$ .

*Challenge phase.* The verifier sends the vector  $\beta$  consisting of  $\lambda$  independently sampled bits to the prover.

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

It follows that, for each  $k = 1, \dots, \lambda$ , the prover sends  $s_{\beta[k],k}$  and  $e_{\beta[k],k}$ , commitments  $c_{s_{1-\beta[k],k}}$  and  $c_{e_{1-\beta[k],k}}$  together with the openings  $d_{s_{\beta[k],k}}$  and  $d_{e_{\beta[k],k}}$ , and the partial decryptions  $\{t_{1-\beta[k],j,k}\}_{j=1}^\tau$ . 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  $c_{s_k} = c_{s_{1-\beta[k],k}} + \text{Com}(s_{\beta[k],k})$  and  $c_{e_k} = c_{e_{1-\beta[k],k}} + \text{Com}(e_{\beta[k],k})$ , with randomness zero, are commitments to  $s_{\beta[k],k} + s_{1-\beta[k],k}$  and  $e_{\beta[k],k} + e_{1-\beta[k],k}$ , 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  $\pi_{S_0}$  and  $\pi_{S_1}$ . Denote all proofs of shortness by  $\pi_S$ .

*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  $\pi_S$ , proving knowledge of valid openings of the commitments  $c_{s_k}$  and  $c_{e_k}$ . We can use the Fiat-Shamir with aborts framework [Lyu09, Lyu12], 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 [BN20] or the range proofs by Attema *et al.* [ALS20]. 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 such as Ligerio [AHIV17] or Aurora [BCR<sup>+</sup>19], adding more complexity to the overall protocol. We can also use the amortized proof by Bootle *et al.* [BBC<sup>+</sup>18] to prove that all  $\lambda$  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  $\lambda$  is much smaller than the number of ciphertexts  $\tau$ , 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.* [KTX08] and Ling *et al.* [LNSW13] 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 [APS15] 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 [Beu20] we can shrink the proofs down to  $18\mu\lambda$  KB. If we replace  $\pi_S$  with the amortized proof by Bootle *et al.* [BBC<sup>+</sup>18] we get a proof of total size 520 KB\*. However, if the number of ciphertexts  $\tau$  is very large, we can ignore all other terms and get a proof of correct decryption  $\pi_D$  of size  $\approx 14\lambda\tau$  KB. See Table 1 for details. The ciphertext modulus  $q$  is chosen to be large enough to ensure correct decryption.

## 7.2 Implementation

We wrote a proof of concept implementation of our scheme in C++ using the NTL-library [Sho21]. 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 NTLlib [ABG<sup>+</sup>16], where they show clear improvements

\* 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 [BBC<sup>+</sup>18, Section 4.1] for details.

\*\* [github.com/tjesi/verifiable-decryption-in-the-head](https://github.com/tjesi/verifiable-decryption-in-the-head).

Parameter	Explanation	Constraints	Value
$N$	Dimension	Power of two	2048
$q$	Ciphertext modulus	$B_{\text{Dec}} \ll q \equiv 1 \pmod{2N}$	$\approx 2^{55}$
$p$	Plaintext modulus		2
$\kappa$	Security parameter	Long-term privacy	128
<b>sec</b>	Statistical security		40
$\lambda$	Soundness parameter		10, ..., 128
$\mu$	Repetitions of $\Pi_{\text{ZKPoS}}$	$\mu \geq \lambda \cdot \ln(2) / \ln(3/2)$	17, ..., 218
$B_\infty$	Bounds on secrets		1
$B_{\text{Dec}}$	Decryption bound	$\ v - su\ _\infty \leq B_{\text{Dec}}$	$\approx 2^{13}$
Size of $\pi_D$	Timings for $\pi_D$	Size of $\pi_S$	Timings for $\pi_S$
$14\lambda\tau$ 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  $\pi_D$  and proofs of shortness  $\pi_S$ .

compared to NTL, indicates that an optimized implementation should provide a speedup by at least an order of magnitude.

## 8 Comparison

### 8.1 Comparison to DistDec (TCC'10)

We sketch an extension of the passively secure distributed decryption protocol  $\Pi_{\text{DistDec}}$  given by Bendlin and Damgård [BD10], which is used in SPDZ [DKL<sup>+</sup>13, DPSZ12]. 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  $\|E_{i,j}\|_\infty \leq 2^{40} B_{\text{Dec}} / \xi p$  (for 40 bits statistical security) and computes the partial decryptions  $t_{i,j} = s_i u_j + p E_{i,j}$  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.* [BDL<sup>+</sup>18].
3. Each party  $\mathcal{D}_i$  use the amortized proof by Baum *et al.* [BBC<sup>+</sup>18] for size  $N$  to prove that each  $E_{i,j}$  is bounded by  $2^{\text{sec}} B_{\text{Dec}} / \xi p$ , for commitments  $\mathbf{c}_{E_{i,j}}$ .
4. The verifier checks the relations  $(v_j - t_{0,j} - t_{1,j} \pmod{q}) \equiv m_j \pmod{p}$  and that all the zero-knowledge proofs are valid.

Elements  $t_j$  and commitments  $\mathbf{c}_{E_{i,j}}$  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  $540 \log(6\bar{\sigma})$

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  $\Pi_{\text{DistDec}}$  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.* [ABGS22] 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  $\Pi_{\text{zkPCD}}$  is of equal size as  $\Pi_{\text{DistDec}}$  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.

## 8.2 Comparison to Boschini *et al.* (PQ Crypto’20)

Boschini *et al.* [BCOS20] 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.

## 8.3 Comparison to Lyubashevsky *et al.* (PKC’21)

A recent publication by Lyubashevsky, Nguyen and Seiler [LNS21] gives a verifiable decryption protocol for the Kyber encapsulation scheme [SAB<sup>+</sup>20]. 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 is 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.

## 8.4 Comparison to Silde (VOTING’22)

Silde [Sil22] presents a direct verifiable decryption of BGV ciphertexts. The parameters are similar to our scheme, and the proof is of 43.6 KB per ciphertext. This is the same as in our scheme for  $\lambda = 3$ , ignoring the setup cost, while smaller for larger  $\lambda$ . The timing of the decryption protocol is 76 ms per ciphertext, which is equal to our timings for  $\lambda = 19$  and otherwise up to 7 times faster for  $\lambda = 128$ .

# 9 Conclusion and Future Work

## 9.1 Summary and Conclusion

We have defined a passively secure distributed decryption protocol, and show how this can be used to construct an interactive zero-knowledge protocol for correct decryption. This is the first both efficient and simple single-party verifiable decryption protocol for lattice-based cryptography when instantiated with the BGV encryption scheme.

The size and efficiency of the protocol is a small factor times  $\lambda\tau$ , for arbitrary soundness parameter  $\lambda$  and number of ciphertexts  $\tau$ . The long-term privacy parameter of the protocol  $\kappa$  can be set independently of, and in particular larger than,  $\lambda$ . This allows an interactive instantiation of the protocol to be very efficient, both in size and computation. For  $\kappa = 128$  we estimate the decryption proof to be of size  $\approx 14\lambda\tau$  KB and the proof/verification time to be only  $4\lambda$  ms per ciphertext, when  $\tau$  is much larger than  $\lambda$ .

Altogether, our new lattice-based proof of decryption provides a unique combination of efficiency and simplicity that make our proof system an interesting candidate for real-world applications.

## 9.2 Future Improvements and Extensions

*Remove the ZK-proofs of shortness.* The Deal-algorithm outputs a zero-knowledge proof proving that the sum of the shares of the secret key and noise used to compute the public key are short. This is to ensure the correctness and security of the encryption scheme. However, ElGamal does not require such a proof, and it might be infeasible to find key-shares that are correct, but not short, that decrypts consistently for all BGV-ciphertexts. We would need to conduct a more careful analysis to ensure that our construction is secure also without the zero-knowledge proofs.

*Instantiations based on other primitives.* A natural future step is to apply our transformation to other encryption schemes, also with other underlying hardness assumptions. As an example, a threshold scheme has been recently constructed based on isogenies [DM20].

## Thanks

We thank Carsten Baum and the anonymous reviewers for helpful comments.



## References

- ABG<sup>+</sup>16. Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrede Lepoint. NFLlib: NTT-based fast lattice library. In Kazue Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 341–356. Springer, Heidelberg, February / March 2016.
- ABGS22. Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. Cryptology ePrint Archive, Report 2022/422, 2022. <https://ia.cr/2022/422>.
- Adi08. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008: 17th USENIX Security Symposium*, pages 335–348. USENIX Association, July / August 2008.
- AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkatasubramanian. Liger: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2087–2104. ACM Press, October / November 2017.
- ALS20. Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. Practical product proofs for lattice commitments. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 470–499. Springer, Heidelberg, August 2020.
- APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
- BBC<sup>+</sup>18. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 669–699. Springer, Heidelberg, August 2018.
- BCHPM04. Yves Bertot, Pierre Castéran, Gérard Huet, and Christine Paulin-Mohring. *Interactive theorem proving and program development : Coq’Art : the calculus of inductive constructions*. Texts in theoretical computer science. Springer, 2004.
- BCOS20. Cecilia Boschini, Jan Camenisch, Max Ovsiankin, and Nicholas Spooner. Efficient post-quantum SNARKs for RSIS and RLWE and their applications to privacy. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 247–267. Springer, Heidelberg, 2020.
- BCR<sup>+</sup>19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, Heidelberg, May 2019.
- BD10. Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio,

- editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 201–218. Springer, Heidelberg, February 2010.
- BDL<sup>+</sup>18. Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 368–385. Springer, Heidelberg, September 2018.
- Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 183–211. Springer, Heidelberg, May 2020.
- BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325. Association for Computing Machinery, January 2012.
- BHLY16. Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 323–345. Springer, Heidelberg, August 2016.
- BKS19. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 3–33. Springer, Heidelberg, May 2019.
- BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 495–526. Springer, Heidelberg, May 2020.
- BS13. Slim Bettaieb and Julien Schrek. Improved lattice-based threshold ring signature scheme. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 34–51. Springer, Heidelberg, June 2013.
- CP92. David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.
- DF90. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO’89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, Heidelberg, August 1990.
- DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*,

- Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122. Springer, Heidelberg, August 2016.
- DKL<sup>+</sup>13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, September 2013.
- DM20. Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 187–212. Springer, Heidelberg, May 2020.
- DOTT21. Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 99–130. Springer, Heidelberg, May 2021.
- DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, Heidelberg, August 2012.
- EFGT17. Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1857–1874. ACM Press, October / November 2017.
- FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO’86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, Heidelberg, August 1987.
- Gor98. Daniel M. Gordon. A Survey of Fast Exponentiation Methods. *J. Algorithms*, 27(1):129–146, 1998.
- HGT19. Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. In *CCS*, pages 685–702. ACM, 2019.
- HLPT20. Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy*, pages 644–660. IEEE Computer Society Press, May 2020.
- HM20. Thomas Haines and Johannes Müller. SoK: Techniques for verifiable mix nets. In Limin Jia and Ralf Küsters, editors, *CSF 2020: IEEE 33rd Computer Security Foundations Symposium*, pages 49–64. IEEE Computer Society Press, 2020.
- HW14. Sven Heiberg and Jan Willemson. Verifiable internet voting in Estonia. In *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014*, 2014.

- IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 21–30. ACM Press, June 2007.
- KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189. Springer, Heidelberg, April / May 2018.
- KTX08. Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 372–389. Springer, Heidelberg, December 2008.
- LM06. Vadim Lyubashevsky and Daniele Micciancio. Generalized compact Knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 144–155. Springer, Heidelberg, July 2006.
- LN16. Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 124–139. Springer, Heidelberg, November 2016.
- LNS21. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 215–241. Springer, Heidelberg, May 2021.
- LNSW13. San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 107–124. Springer, Heidelberg, February / March 2013.
- LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, Heidelberg, May / June 2010.
- LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 35–54. Springer, Heidelberg, May 2013.
- LW18. Fucui Luo and Kunpeng Wang. Verifiable decryption for fully homomorphic encryption. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *ISC 2018: 21st International Conference on Information Security*, volume 11060 of *Lecture Notes in Computer Science*, pages 347–365. Springer, Heidelberg, September 2018.

- Lyu09. Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer, Heidelberg, December 2009.
- Lyu12. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755. Springer, Heidelberg, April 2012.
- PBD07. Kun Peng, Colin Boyd, and Ed Dawson. Batch zero-knowledge proof and verification and its applications. *ACM Trans. Inf. Syst. Secur.*, 10(2):6, 2007.
- SAB<sup>+</sup>20. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- Sho21. Victor Shoup. Ntl: A library for doing number theory, 2021. <https://libnt1.org/index.html>.
- Sil22. Tjerand Silde. Verifiable Decryption for BGV. Workshop on Advances in Secure Electronic Voting, 2022. <https://ia.cr/2021/1693>.
- SSA<sup>+</sup>18. Fatemeh Shirazi, Milivoj Simeonovski, Muhammad Rizwan Asghar, Michael Backes, and Claudia Diaz. A survey on routing in anonymous communication protocols. *ACM Comput. Surv.*, 51(3), June 2018.
- Ste94. Jacques Stern. A new identification scheme based on syndrome decoding. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO’93*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer, Heidelberg, August 1994.