





Verifiable Decryption in the Head

Kristian Gjøsteen¹ , Thomas Haines¹, Johannes Müller² ,
Peter Rønne² , and Tjerand Silde¹ 

¹ Norwegian University of Science and Technology
{kristian.gjosteen,thomas.haines,tjerand.silde}@ntnu.no

² University of Luxembourg
{johannes.mueller,peter.roenne}@uni.lu

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 first present a toy example for an ElGamal distributed decryption protocol before applying our method to a lattice-based scheme. This leads to an efficient lattice-based verifiable decryption with only one server; it has lightweight computations as we reduce the need of zero-knowledge proofs. 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 where the online phase can be very efficient.

Keywords: Verifiable decryption · MPC in the head · Passively secure distributed decryption · Lattice-based cryptography.

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⁺18], decryption of ballots in electronic voting [HM20], and various uses of 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 preprint by Lyubashevsky *et al.* [LNS20] (which provides a rather complicated decryption proof consisting of a mix of proofs of linear relations, proofs of shortness and range proofs), no efficient and straight-forward zero-knowledge proofs (ZKPs) 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 ZKPs of correct decryption should also be secure

in the face of quantum computers which are becoming increasingly more powerful. For example, the electronic voting systems Helios [Adi08], Scytl [Scy18] or 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 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 [BLO18, BD10, BKP13, BS13, BGG⁺18, DPSZ12, DOTTT20]. 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 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 the decryption 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 El-Gamal, 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 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 both shares of the decryption key, any auxiliary information related to them, and decryption shares. Again, 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 the 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.

As part of our construction, we also design a suitable instantiation of a 2-party passively secure distributed decryption scheme for a standard lattice-based encryption scheme. As discussed above, there are several obstacles that need to be solved, some of which are fairly simple. For the remaining we use existing lattice-based arguments and commitments.

Even though we use existing lattice-based arguments (which, in principle, could also be used to do a decryption proof directly), we only use these arguments once per run of the 2-party distributed decryption protocol, which means that the number of arguments is linear in the security parameter, not linear in the number of ciphertexts. We use an amortized proof here, for even greater speedup.

Combined with the main contribution, this gives us a verifiable decryption scheme for a lattice-based public key encryption scheme that is very fast when the number of ciphertexts is much larger than the security parameter. The proof size is also quite reasonable, in particular for an interactive proof with lower security parameter.

1.2 Related Work

Verifiable decryption for ElGamal can be done by proving the equality of two discrete logarithms [CP92], which is well understood, fairly cheap on its own, and can also be batched for significantly improved performance when decrypting many ciphertexts [Gor98, PBD07].

The "dual" Regev system, see e.g. [GPV08, Section 7.1] or [LPR13, Section 8.1], can be used for verifiable decryption 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). Boschini *et al.* [BCOS20] provide a verifiable decryption scheme which offers good communication complexity, but the running time of the protocol would take several minutes per ciphertexts, and would thus be impractical for all but very small sets of ciphertexts. In a recent preprint, Lyubashevsky *et al.* [LNS20] provides a very efficient protocol for small message spaces, while the proof is comparable to ours for larger parameters. Furthermore, there is a line of new results for proving shortness of lattice-based vectors in zero-knowledge. These protocols offer practical sizes and timings. We discuss them in more details in Section 8.

Threshold encryption schemes [DF90] and distributed decryption schemes are now well-understood, and many constructions exist [BD10], in particular those related to SPDZ [BCS19, DKL⁺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. Since our approach gives away a decryption key share and any randomness involved, it is trivial to verify that the key share has been used correctly, allowing us to avoid this issue.

1.3 Paper Overview

We define the variant of distributed decryption schemes that we need in Section 2, along with the required security notions. The main difference from the usual definition is that the key generation algorithm outputs a single decryption key, not a list of decryption key shares. Instead, we have a dealer algorithm that outputs decryption key shares and some auxiliary data that allows a verification

algorithm to check that a given decryption key share is correct. Also, there is no algorithm to verify that a decryption share is correct, since we verify this by revealing the decryption key share and any randomness used.

We describe our main contribution in Section 3: a verifiable decryption scheme based on our variant of distributed decryption. For ease of understanding, we have included a toy example based on ElGamal. We note that this toy example does not make practical sense since there are much faster solutions available for ElGamal, but the example is included as an aid to the reader.

Section 4 provides background on machine checked proofs and details our the results of machine checked proofs.

Section 5 covers the needed background on lattice-based cryptography, including the underlying encryption scheme, a commitment scheme and some proofs that we need. Section 6 then covers the construction of our threshold public key encryption scheme and the resulting verifiable decryption scheme.

We suggest parameters and carefully analyze the performance of the verifiable decryption scheme in Section 7. We also compare our construction to a verifiable decryption scheme derived from an hypothetical variant of a threshold decryption scheme [BD10, DKL⁺13, DPSZ12] and some other protocols in Section 8.

2 Passively Secure 2-party Distributed Decryption

A *distributed decryption scheme* enables a set of players to distribute the decryption of ciphertexts, in such a way that only allowed subsets of players can do the decryption. Usually, the decryption key shares are created once during key generation. As discussed in the introduction, we shall need to 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 a key generation algorithm **KeyGen**, an encryption algorithm **Enc** and a decryption algorithm **Dec**. We extend the notation with a predicate **KeyMatch** which takes as input a public and secret key. We require that for all matching public and secret keys pk, sk and all messages m , we have $\text{Dec}(\text{sk}, \text{Enc}(\text{pk}, m)) = m$ (at least 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. For the purpose of this work, we consider only two parties where both are required to decrypt.

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

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*.

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

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

Definition 1 (Correctness). *A distributed decryption protocol is correct if for all ciphertexts c , any key pair (pk, sk) s.t. $\text{KeyMatch}(pk, sk) = 1$, any (sk_0, sk_1, aux) output by $\text{Deal}(pk, sk)$, then $\text{Verify}(pk, aux, i, sk_i) = 1$, for $i = 0, 1$, and*

$$\Pr [m \leftarrow \text{Dec}(sk, c); \text{Reconstruct}(c, \text{Play}(sk_0, c), \text{Play}(sk_1, c)) = m] \geq 1 - \text{negl}.$$

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 `Verify` algorithm), then the `Play` and `Reconstruct` 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 ciphertexts c , public keys pk , secret key shares (sk_1, sk_2) , and auxiliary data aux and sk output by $\text{FindKey}(pk, sk_0, sk_1, aux)$ satisfying $\text{Verify}(pk, aux, i, sk_i) = 1$, for $i = 0, 1$, we have that*

$$\Pr [\text{KeyMatch}(pk, sk) \wedge \text{Reconstruct}(c, \text{Play}(sk_0, c), \text{Play}(sk_1, c)) = \text{Dec}(sk, c)] \geq 1 - \text{negl}.$$

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.

$\frac{\text{Exp}_{\mathcal{A}}^{\text{ddp-sim-0}}(pk, sk)}{(i, (c_0, \dots, c_\tau), (m_0, \dots, m_\tau)) \leftarrow \mathcal{A}(pk)}$ $(sk_0, sk_1, aux) \leftarrow \text{Deal}(pk, sk)$ $\forall j: ds_j \leftarrow \text{Play}(sk_{1-i}, c_j)$ $b = \mathcal{A}(aux, sk_i, (ds_0, \dots, ds_\tau))$ <p>return b</p>	$\frac{\text{Exp}_{\mathcal{A}}^{\text{ddp-sim-1}}(pk)}{(i, (c_0, \dots, c_\tau), (m_0, \dots, m_\tau)) \leftarrow \mathcal{A}(pk)}$ $(sk_i, aux) \leftarrow \text{DealSim}(pk, i)$ $\forall j: ds_j \leftarrow \text{PlaySim}(pk, sk_i, c_j, m_j)$ $b = \mathcal{A}(aux, sk_i, (ds_0, \dots, ds_\tau))$ <p>return b</p>
--	--

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

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

where the probability is taken over the random tapes and (pk, 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 .

2.1 Toy Example: Distributed Decryption for ElGamal

We briefly recall ElGamal encryption for a given cyclic group \mathbb{G} of prime order p with generator \mathbf{g} .

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

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

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 (KeyMatch) takes as input a public key $\text{pk} \in \mathbb{G}$ and a secret key $x \in \mathbb{Z}_p^*$ and returns $\mathbf{g}^x = \text{pk}$.

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 $(x_0, x_1, \text{aux} = (\mathbf{g}^{x_0}, \mathbf{g}^{x_1}))$.

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

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

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

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

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

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 by the same easy computations as above.

Privacy. The simulators DealSim and PlaySim work as follows:

- DealSim takes the public key pk and a bit i , samples x_i from \mathbb{Z}_p^* and returns (wlog.) $(x_i, (\mathbf{g}^{x_i}, \text{pk}/\mathbf{g}^{x_i}))$. 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 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 (see notes by Damgård [Dam10] or Krenn and Orrù [KO21] for a detailed introduction to sigma protocols).

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. (Traditionally, decryption failures were never allowed. Recent cryptosystems have a very small fraction of decryption failures.)

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 result of decrypting 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 then sends the auxiliary information from Deal and all the decryption shares to the verifier. After that, the verifier returns a challenge which is a binary vector of length λ . The prover then reveals the corresponding parts of the shares, as well as any randomness used in the Play algorithms with this key share. The prover

now 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.

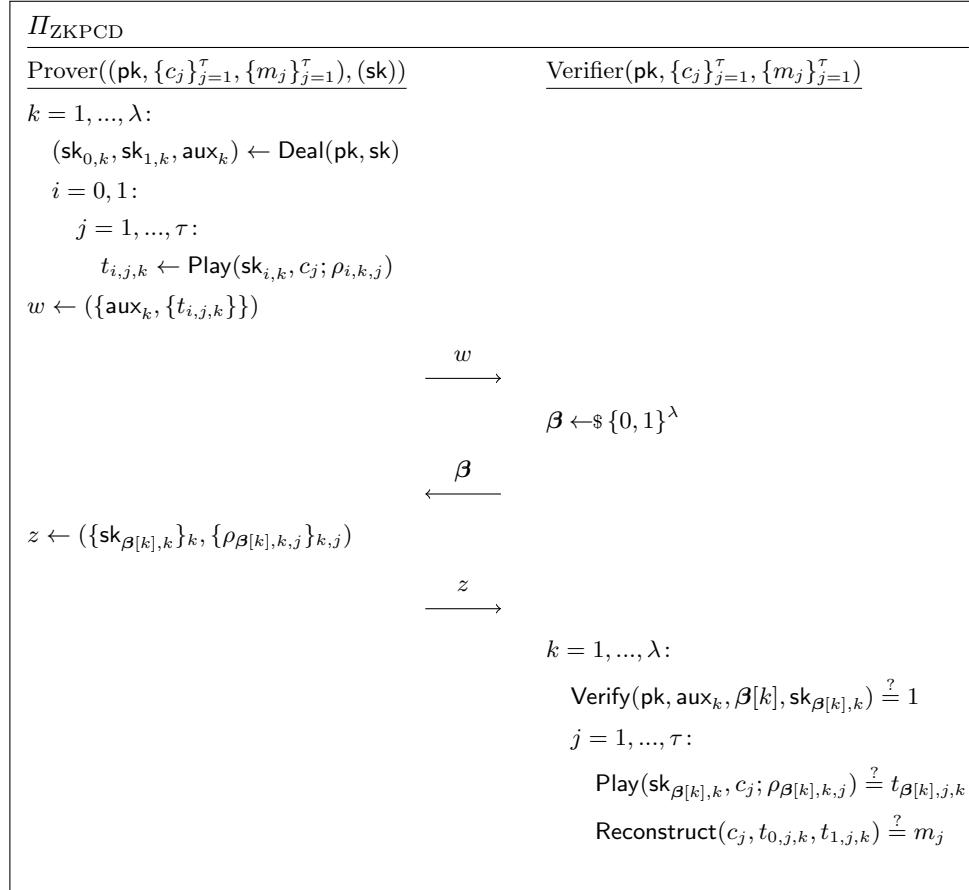


Fig. 2. Proof of correct decryption. Here, $\rho_{i,k,j}$ denotes the random tape used by the Play algorithm to create the i th share of the j th ciphertext in the k th run of the distributed decryption scheme.

Including the random tapes used by the Play algorithm is necessary, because this algorithm may be randomized. However, in our toy example (Sec. 2.1), the Play algorithm was deterministic, so there was no randomness involved. For our lattice-based algorithm, it turns out that we can verify the correctness of the decryption share by other means, so again there is no need to transmit the random tape.

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.

Soundness. By rewinding, any cheating prover with a significant success probability can be used to create two accepting conversations (w, β, z) and (w, β', 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 β : 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 $\text{ds}_{\beta[i],j,i} \leftarrow \text{PlaySim}(\text{pk}, \text{sk}_{\beta[i],i}, c_i, m_i)$ and $\text{ds}_{1-\beta[i],j,i} \leftarrow \text{Play}(\text{pk}, \text{sk}_{\beta[i],i}, c_i)$. 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 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 files can be found at www.dropbox.com/s/mn9gfmw3utkffiyq.

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

We start by giving the preliminaries on cyclotomic rings, the discrete Gaussian distribution and hardness assumptions based on knapsacks. We continue by presenting the BGV encryption scheme [BGV12] and a commitment scheme, both with security based on the hardness of the given knapsack-problems. Lastly we describe the amortized zero-knowledge proof of knowledge protocol by Bootle *et al.* [BLNS20] for proving knowledge of many short openings given a fixed public commitment matrix.

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, LMPR08].

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 Discrete Gaussian Distribution

The continuous normal distribution ρ over \mathbb{R}^k , k -dimensional vectors over the real numbers, centered at $\mathbf{v} \in \mathbb{R}^k$ with standard deviation $\sigma \in \mathbb{R}^{>0}$, is given by

$$\rho(\mathbf{x})_{\mathbf{v},\sigma}^k = \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^k \exp\left(\frac{-\|\mathbf{x} - \mathbf{v}\|_2^2}{2\sigma^2} \right).$$

The *discrete Gaussian distribution* \mathcal{N} over the ring R is achieved by normalizing the continuous distribution over R , for $\mathbf{x} \in R^k$ represented as a kN -dimensional vector, by letting

$$\mathcal{N}_{\mathbf{v},\sigma}(\mathbf{x}) = \frac{\rho_{\mathbf{v},\sigma}^{kN}(\mathbf{x})}{\rho_{\sigma}^{kN}(R)},$$

where $\rho_\sigma^{kN}(R) = \sum_{\mathbf{x} \in R} \rho_\sigma^{kN}(\mathbf{x})$. When $\sigma = 1$ or $\mathbf{v} = \mathbf{0}$, they are omitted. When \mathbf{x} is sampled according to \mathcal{N}_σ (see Section 2.1 in [BBC⁺18]), then, for $\gamma \in \mathbb{R}^{>0}$,

$$\Pr[\|\mathbf{x}\|_\infty > \gamma\sigma] \leq 2e^{-\gamma^2/2} \quad \text{and} \quad \Pr[\|\mathbf{x}\|_2 > \sqrt{2\gamma}\sigma] < 2^{-\gamma/4}.$$

5.3 Knapsack Problems

We first define the Search Knapsack problem in the ℓ_2 norm, also denoted as SKS². The SKS² problem is exactly the Ring-SIS problem in its Hermite Normal Form, see [PR06, LM06, LPR10] for more details.

Definition 4. *The SKS²_{N,q,β} problem is to find a short vector \mathbf{y} in R_q^2 satisfying $[a' \ 1] \cdot \mathbf{y} = 0$ for a given random a' in R_q . An algorithm \mathcal{A} has advantage ϵ in solving the SKS²_{N,q,β} problem if*

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

Additionally, we define the Decisional Knapsack problem in the ℓ_∞ norm denoted as DKS[∞]. The DKS[∞] problem is equivalent to the Ring-LWE problem when the number of samples is limited.

Definition 5. *The DKS[∞]_{N,q,β} problem is to distinguish the distribution $[a' \ 1] \cdot \mathbf{y}$, for a short \mathbf{y} , from the uniform distribution when given random a' in R_q . An algorithm \mathcal{A} has advantage ϵ in solving the DKS[∞]_{N,q,β} problem if*

$$\begin{aligned} & |\Pr[b = 1 \mid a' \leftarrow \$ R_q; \mathbf{y} \leftarrow \$ R_q^2 \text{ s.t. } \|\mathbf{y}\|_\infty \leq \beta; b \leftarrow \mathcal{A}(a', [a' \ 1] \cdot \mathbf{y})] \\ & - \Pr[b = 1 \mid a' \leftarrow \$ R_q; u \leftarrow \$ R_q; b \leftarrow \mathcal{A}(a', u)]| \geq \epsilon. \end{aligned}$$

5.4 BGV Encryption

We present the BGV encryption scheme by Brakerski, Gentry and Vaikuntanathan [BGV12]. Let $p \ll q$ be primes, let R_q and R_p be defined as above for a fixed N , let \mathcal{N}_σ be a Gaussian distribution over R_q with standard deviation σ , let $B_\infty \in \mathbb{N}$ be a bound and let κ be the security parameter. These are the public parameters pp . The BGV encryption scheme consists of three algorithms: key generation (**KeyGen**), encryption (**Enc**) and decryption (**Dec**), where

- **KeyGen** samples an element $a \leftarrow \$ R_q$ uniformly at random, samples a short $s \leftarrow \$ R_q$ such that $\|s\|_\infty \leq B_\infty$ and samples noise $e \leftarrow \mathcal{N}_\sigma$. The algorithm outputs public key $\text{pk} = (a, b) = (a, as + pe)$ and secret key $\text{sk} = (s, e)$.
- **Enc**, on input the public key $\text{pk} = (a, b)$ and an element m in R_p , samples a short $r \leftarrow \$ R_q$ such that $\|r\|_\infty \leq B_\infty$, samples noise $e', e'' \leftarrow \mathcal{N}_\sigma$, and outputs the ciphertext $c = (u, v) = (ar + pe', br + pe'' + m)$.

- **Dec**, on input the secret key $\mathbf{sk} = s$ and a ciphertext $c = (u, v)$ in R_q^2 , outputs the message $m \equiv (v - su \bmod q) \bmod p$.

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

Furthermore, we present the passively secure distributed decryption technique used in the MPC-protocols by Damgård *et al.* [BD10,DKL⁺13,DPSZ12]. When decrypting, we assume that each decryption server \mathcal{D}_j , for $1 \leq j \leq \xi$, has a uniformly random share $\mathbf{sk}_j = s_j$ of the secret key $\mathbf{sk} = s$ 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 $\mathbf{sk}_j = s_j$ and a ciphertext $c = (u, v)$ in R_q^2 , computes $m_j = s_j u$, sample some large noise $E_j \leftarrow_{\$} R_q$ such that $\|E_j\|_\infty \leq 2^{\text{sec}}(B_{\text{Dec}}/p\xi)$ for some statistical security parameter **sec** and upper error-bound $\max\|v - su\|_\infty \leq B_{\text{Dec}}$, then outputs $t_j = m_j + pE_j$.

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 $\|v - t\|_\infty \leq (1 + 2^{\text{sec}})B_{\text{Dec}} < \lfloor q/2 \rfloor$ (see [DKL⁺13, Appendix G]). Here, t will be indistinguishable from random except with probability $2^{-\text{sec}}$.

5.5 Lattice-Based Commitments

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 and the bound on s , the value $b = as + e$ is a commitment to an arbitrary secret s with secret randomness e – if e is short and a is a uniformly random public element.

More formally, 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 **pp**. The commitment scheme consists of three algorithms: key generation (**KeyGen**), commit (**Com**) and open (**Open**), where

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

It follows that the commitment scheme is hiding if the $\text{DKS}_{N,q,B'_\infty}^\infty$ -problem is hard and that it is binding if the $\text{SKS}_{N,q,B'_\infty \cdot \sqrt{N}}^2$ -problem is hard.

5.6 Exact Amortized Zero-Knowledge Proof of Short Openings

Bootle *et al.* [BLNS20] give an efficient amortized zero-knowledge protocol for proving the knowledge of short vectors \mathbf{s}'_i and \mathbf{e}'_i over \mathbb{Z}_q^N satisfying $\mathbf{A}'\mathbf{s}'_i + \mathbf{e}'_i = \mathbf{u}'_i$. Here, \mathbf{A}' is a fixed public matrix in $\mathbb{Z}_q^{N \times N}$ and \mathbf{u}'_i are public commitments in \mathbb{Z}_q^N . By “short” we mean that $\|\mathbf{s}'_i\|_\infty$ and $\|\mathbf{e}'_i\|_\infty$ are bounded by some bound B' , and we present the protocol for $B' = 1$, that is, all coefficients of \mathbf{s}'_i and \mathbf{e}'_i are ternary. The protocol is described in detail in Figure 3.

We note that all polynomials in R_q can be represented as vectors in \mathbb{Z}_q^N and that multiplication by a polynomial a' in R_q can be represented as matrix-vector product with a negacyclic matrix \mathbf{A}' in $\mathbb{Z}_q^{N \times N}$. It follows that we can use this protocol to prove, in zero-knowledge, that a commitment c_m is a valid commitment to a ring element with short entries.

Let \mathbb{H} be a hash function with 2κ bits output. We provide a short explanation of the different parts of the protocol, and refer to [BLNS20] for more details:

- We prove that $\mathbf{A}'\mathbf{s}'_i + \mathbf{e}'_i = \mathbf{u}'_i$ for $i = 1, \dots, \lambda$ and that \mathbf{s}'_i and \mathbf{e}'_i are ternary.
- Vector \mathbf{s}'_0 is sampled to hide the secrets $\mathbf{s}'_{i>0}$ and to ensure zero-knowledge.
- Polynomials $\ell_i(X)$ are Lagrange interpolation polynomials so that $\ell_i(X) = \prod_{j \neq i} (X - a_j) / (a_i - a_j)$ for fixed a_1, \dots, a_λ distinct interpolation points in \mathbb{Z}_q .
- We let \circ denote coordinate-wise multiplication of elements of two vectors.
- If \mathbf{s}'_i is ternary, then the products $s'_{i,j}(s'_{i,j} - 1)(s'_{i,j} + 1)$ are all equal to zero. It follows that the constant term in the product involving \mathbf{f}' will be zero.
- It also follows that the constant term in the product involving \mathbf{d}' is zero.
- Let \mathcal{RS} be a Reed-Solomon code with encoding function **Encode**. The inputs are vectors over \mathbb{Z}_q of length $k = N + 7\eta$ and the outputs are vectors over \mathbb{Z}_q of length l , for $k < l < q$. The minimum distance of the code is $d = l - k + 1$.
- Adding randomness $\mathbf{r}'_{i,j}$ of dimension η information theoretically hides all information about the input when revealing codewords in up to η positions.
- Let **RowsToMatrix** denote the function taking as input an ordered list of row-vectors and outputs a matrix consisting of the input rows in order.
- Let **MerkleTree** be a function that takes as input a matrix and outputs a root of a Merkle tree with the columns of the matrix as leaf nodes.
- Let **MerklePaths $_I$** be the set of hashes along the paths from leaf nodes with index in I , proving that a given set of elements is in the tree.
- To compress the size of the paths in the Merkle trees we split it into h trees with l/h leafs each.
- Let **Verify** be a function that takes as input a set of column vectors, a Merkle root and a set of paths, and verifies that each column is a leaf node.

The verification equation is the following:

$$\text{Encode} \left(\bar{\mathbf{f}}', \bar{\mathbf{f}}' \circ [\bar{\mathbf{f}}' - \mathbf{1}] \circ [\bar{\mathbf{f}}' + \mathbf{1}], \bar{\mathbf{d}}' \circ [\bar{\mathbf{d}}' - \mathbf{1}] \circ [\bar{\mathbf{d}}' + \mathbf{1}], \bar{\mathbf{r}}' \right) \Big|_I \stackrel{?}{=} \sum_{i=0}^{\lambda} \ell_i(x) \mathbf{H}_i \Big|_I. \quad (1)$$

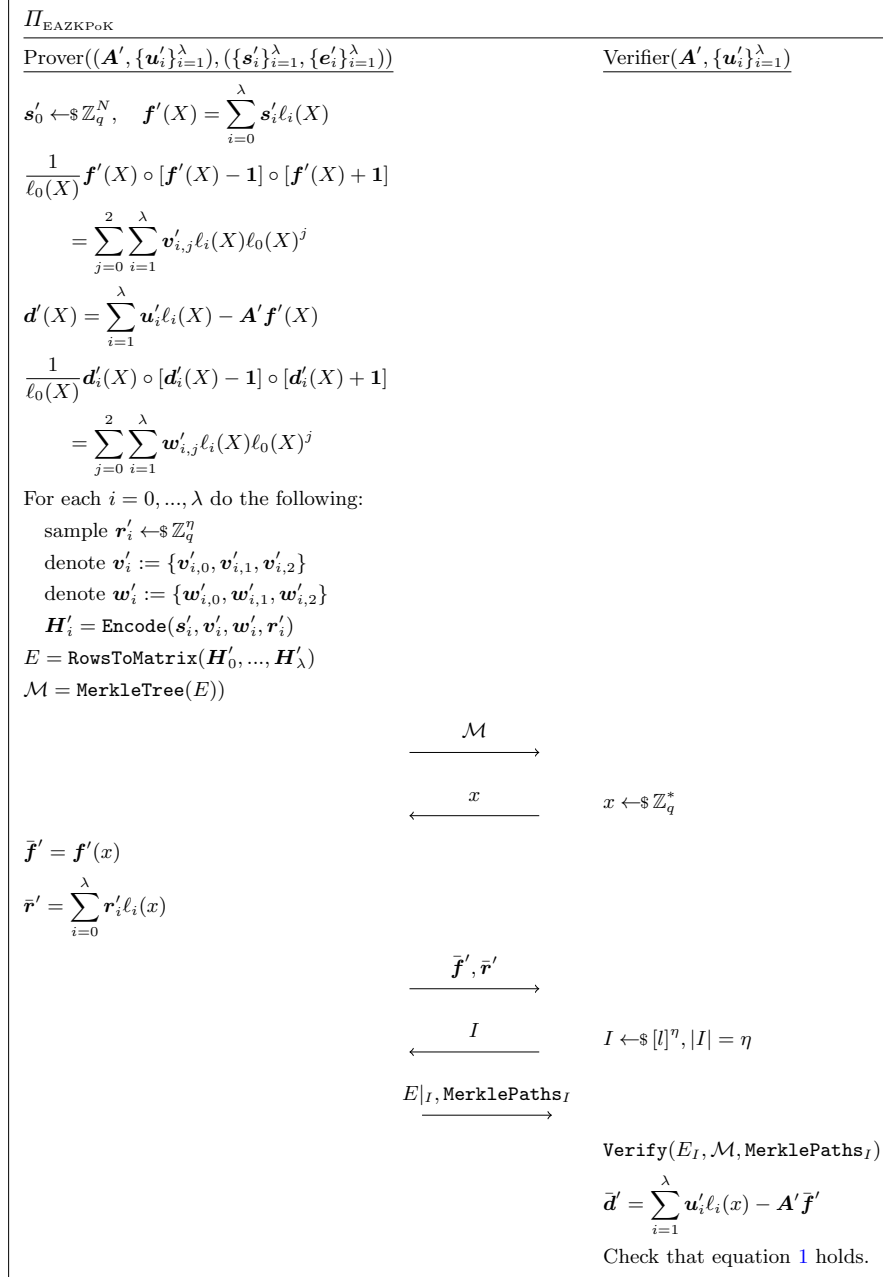


Fig. 3. The exact amortized zero-knowledge proof of knowledge of short openings.

It is straightforward to see that the protocol is complete and zero-knowledge. The following theorem describe the soundness of the protocol.

Lemma 1 ((Adapted) Theorem 4.3 in [BLNS20]). *Let \mathcal{RS} be the Reed-Solomon code above of dimension $k = N + 7\eta$ and length l with $k < l < q$. Then, the soundness of the protocol in Figure 3 is bounded by*

$$\max \left\{ 2 \left(\frac{k}{l - \eta} \right)^\eta, \frac{8\lambda}{q} \right\}.$$

Furthermore, the size of the amortized zero-knowledge proof in Figure 3 is:

$$(N + (\lambda + 2)\eta) \log q + 2\kappa(h + \eta \log(l/h)) \text{ bits.}$$

The protocol can be made non-interactive with the Fiat-Shamir transform [FS87], and we denote by

$$\begin{aligned} \pi_A &\leftarrow \Pi_{\text{EAZKPoK}}(\mathbf{A}', \{\mathbf{u}'_i\}_{i=1}^\lambda, (\{\mathbf{s}'_i\}_{i=1}^\lambda, \{\mathbf{e}'_i\}_{i=1}^\lambda)), \text{ and} \\ 0 \vee 1 &\leftarrow \Pi_{\text{EAZKPoKV}}(\mathbf{A}', \{\mathbf{u}'_i\}_{i=1}^\lambda, \pi_A), \end{aligned}$$

the run of the prover and the verification protocols, respectively.

6 Zero-Knowledge Protocol of Correct Decryption

6.1 Lattice-Based Threshold Decryption Scheme

Setup. We will be working over the cyclotomic ring $R_q = \mathbb{Z}_q[X]/\langle X^N + 1 \rangle$ as defined in Section 5.1, 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 , statistical security parameter sec , standard deviation σ , bound B_∞ and maximal ciphertext error-bound B_{Dec} . Let \mathbb{E} denote the set $\{E \in R_q : \|E\|_\infty \leq 2^{\text{sec}-1} B_{\text{Dec}}/p\}$.

Scheme. We present a threshold decryption version of the BGV encryption scheme [BGV12], where KeyGen , Enc and Dec are defined in Section 5.4.

The dealer algorithm (Deal) takes as input a public key $\text{pk} = (a, b)$ and corresponding secret key $\text{sk} = (s, e)$, 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 a non-interactive zero-knowledge proof π_S proving that the sum $e_0 + e_1$ is short, as described below. It outputs key shares $sk_0 = (s_0, e_0)$, $sk_1 = (s_1, e_1)$ and $\text{aux} = (b_0, b_1, c_{s_0}, c_{s_1}, c_{e_0}, c_{e_1}, \pi_S)$.

The verify algorithm (Verify) takes as input a public key $\text{pk} = (a, b)$, an index i , a secret key share $\text{sk}_i = (s_i, e_i)$ and aux . It outputs 1 if and only if $(b_i \stackrel{?}{=} as_i + pe_i) \wedge (b \stackrel{?}{=} b_0 + b_1) \wedge (\Pi_{\text{Verify}}(c_{e_0}, c_{e_1}, \pi_S))$, and 0 otherwise.

The player algorithm (*Play*) takes as input a key share $\text{sk}_i = (s_i, e_i)$ and a ciphertext $c = (u, v)$, samples uniform E_i in \mathbb{E} and outputs $t_i = s_i u + p E_i$.

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

Optimized Zero-Knowledge Proof of Shortness. There are many options for π_S , proving that the sum of the underlying values of the commitments c_{e_0} and c_{e_1} are short. 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. We can achieve smaller slack using 0/1-challenges in a commit-and-prove protocol, but we would then have to run the protocol many times to achieve negligible soundness. Other alternatives are the exact proofs by Beullens [Beu20] or Baum and Nof [BN20], where proof sizes would be a several hundred kilobytes each, or the more efficient range proofs by Attema *et al.* [ALS20]. However, assuming that we will run the Deal-algorithm λ times, we will use the amortized proof by Bootle *et al.* [BBC⁺18] described in Section 5.6 to prove that all λ executions are done correctly at the same time.

6.2 Security of the Threshold Decryption Scheme

Theorem 1 (Correctness). *The lattice-based threshold decryption scheme defined in Section 6.1 is correct with respect to Definition 1.*

Proof. It follows directly that the *Verify*-algorithm is correct from the correctness of the encryption scheme, commitment scheme and the zero-knowledge protocol defined in Section 5. It also follows that the *Reconstruct*-algorithm is correct from the correctness of the distributed decryption algorithm defined in Subsection 5.4, except with negligible probability over the randomness used to generate the ciphertexts. \square

Theorem 2 (Integrity). *The lattice-based threshold decryption scheme defined in Section 6.1 has integrity with respect to Definition 2.*

Proof. For *Verify* to accept for both $i = 0$ and $i = 1$, we need that $b = b_0 + b_1$, $b_0 = a s_0 + p e_0$, $b_1 = a s_1 + p e_1$ and that the zero-knowledge proof of shortness π_S of the sum $e_0 + e_1$ is accepted. If either of the key shares are incorrect then *Verify* accept with probability 0, and if the key shares are correct, then *Reconstruct* 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 sum $e_0 + e_1$ is not short. The soundness of *Verify* then reduces to the soundness of the zero-knowledge protocol, which has negligible soundness error.

An attacker can also try to find $\hat{s} \neq s$ and short $\hat{e} \neq e$ such that $b = a s + p e = a \hat{s} + p \hat{e}$, that is, he finds a short solution to a knapsack problem, and hence, he breaks SKS^2 -assumption. We conclude that if the SKS^2 -problem is hard and the

zero-knowledge protocol is secure, then the lattice-based threshold decryption scheme has integrity. \square

Theorem 3 (Privacy). *The lattice-based threshold decryption scheme defined in Section 6.1 is, using a simulator $\text{Sim}_{\text{short}}$ for proof of shortness π_S , simulatable with respect to Definition 3. We present simulator DealSim in Figure 4 and simulator PlaySim in Figure 5.*

Proof. We present first a simulator DealSim for the Deal -algorithm and then a simulator PlaySim for the Play -algorithm. We replace all parts of the transcript with uniformly random elements from the same sets and ignore unopened values.

<pre> DealSim(pk = (a, b), i) ----- (b₀[*], b₁[*]) ← ⟨b⟩ s_i[*] ←^{\$} R_q, e_i[*] ≡_q (b_i[*] - as_i[*])/p c_{s_i}[*] ← Com(s_i[*]), c_{s_{1-i}}[*] ← Com(0) c_{e_i}[*] ← Com(e_i[*]), c_{e_{1-i}}[*] ← Com(0) π_S[*] ← Sim_{Short}(c_{e_i}[*], c_{e_{1-i}}[*]) aux[*] ← (b₀[*], b₁[*], c_{s₀}[*], c_{s₁}[*], c_{e₀}[*], c_{e₁}[*], π_S[*]) return (sk_i[*] = (s_i[*], e_i[*]), aux[*]) </pre>
--

Fig. 4. Simulator DealSim .

DealSim : we first note that because we can simulate π_S , we can use $\text{Sim}_{\text{Short}}$ to simulate a proof π_S^* . This allows us to commit to any 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.

PlaySim : we start by computing $t_i = us_i$. Then we find a t_{1-i} such that $(v - t_i - t_{1-i} \bmod q) \bmod p = m$. This ensures that Reconstruct outputs the message m when reconstructing the shares. \square

6.3 Zero-Knowledge Proof of Verifiable Decryption

We will now present the different phases of our sigma protocol for proving correct decryption. The full protocol is given in Figure 6. The security of the construction follows from the results in Section 3 in combination with Theorem 1, 2 and 3.

$\text{PlaySim}(\text{pk} = (a, b), \text{sk}_i = (s_i, e_i), c = (u, v), m)$ <hr style="border: 0.5px solid black;"/> $E^* \xleftarrow{\$} \mathbb{E}, \quad t_i = us_i$ $x_{1-i}^* = v - m_j - t_i$ $t_{1-i}^* = x_{1-i}^* + pE^*$ $\text{return } (ds = t_{1-i}^*)$
--

Fig. 5. Simulator PlaySim.

The reader should note that there are some differences with the abstract construction in Section 3. First, we do not transmit the randomness used. Instead, the verifier reconstructs the randomness and verifies that it is sufficiently short. However, to optimize the size of the protocol, this will change in the next section, using classical commit-and-prove techniques. Second, we do not prove the decryption key shares correct using individual proofs. Instead, we use an amortized proof. This is an optimization, and does not affect the security proof.

Setup. We are given a honestly generated public key $\text{pk} = (a, b = as + pe)$, where $\|s\|_\infty \leq B_\infty$ and $e \leftarrow \mathcal{N}_\sigma$. The secret key $\text{sk} = (s, e)$ is given to the prover. Additionally, we are given a set of properly generated ciphertexts $\{(u_j, v_j) = (ar_j + pe'_j, br_j + pe''_j + m_j)\}_{j=1}^\tau$, where $\|r_j\|_\infty \leq B_\infty$ and $e'_j, e''_j \leftarrow \mathcal{N}_\sigma$, together with a set of messages $\{m_j\}_{j=1}^\tau$ for $m_j \in R_p$. The goal of the prover is to convince the verifier that it knows the secret key and that the set of messages are proper decryptions of the ciphertexts with respect to the given public key.

Commit phase. For a given soundness parameter λ , the prover will do the following for $k = 1, \dots, \lambda$. First, it runs the Deal algorithm on sk and pk to produce $\text{sk}_{0,k}, \text{sk}_{1,k}$ and aux_k . Then, for $i = 0, 1$ and each $j = 1, \dots, \tau$, it runs the Play algorithm on each key-share $\text{sk}_{i,k}$ and ciphertext c_j to produce $t_{0,j,k}$ and $t_{1,j,k}$. Finally, it sends $w \leftarrow (\{\text{aux}_k, \{t_{i,j,k}\}_{i=0,j=1}^{1,\tau}\}_{k=1}^\lambda)$ to end the commitment phase. Note that we can prove that all sharings of e are short because the commitment scheme is slightly additive homomorphic, allowing us to prove shortness of sums of commitments with the same index k .

Challenge phase. The verifier samples a random binary challenge vector β of length λ , where each entry is sampled independently. It sends β to the prover.

Respond phase. The prover sends the openings $(\{d_{s_{\beta[k],k}}, d_{e_{\beta[k],k}}\}_{k=1}^\lambda)$, for each of the corresponding commitments to each index k of β , 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 π_A is valid. Further, for each $j = 1, \dots, \tau$, the verifier runs the **Reconstruct** algorithm to make sure that all decryption shares are correct and that all messages are decrypted correctly. Finally, 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 λ initially can be very small in the interactive case, while it should be (almost) as large as the security parameter κ in the non-interactive setting, increasing the size of the proof of 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 Fig. 6. 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. Also, we use pseudo-randomness to compress the randomness used by the Play algorithm, and point out some further improvements.

7.1 Proof Size

Each element in R_q is of size $N \log q$ bits, which might be large. We will replace some of them with hashes or seeds, and let the verifier re-construct the ring elements whenever possible. We model the hash-function as a random oracle with output length 2κ and denote it by H . Note that the soundness parameter λ of the sigma-protocol may be chosen independently of the long-term 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, 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 β consisting of λ bits to the prover.

Respond phase. First, 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 $E_{\beta[k],j,k}$. 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 $b_{\beta[k],k}$. Instead of sending both $s_{\beta[k],k}$ and $e_{\beta[k],k}$, we can let $s_{\beta[k],k}$ and $e_{1-\beta[k],k}$ be generated from a seed if $\beta[k]$ is zero (and compute $s_{1-\beta[k],k}$ from s and $e_{1-\beta[k],k}$ from e), and opposite when $\beta[k]$ is one. This saves one ring element per round. Lastly, we assume that a uniformly random seed

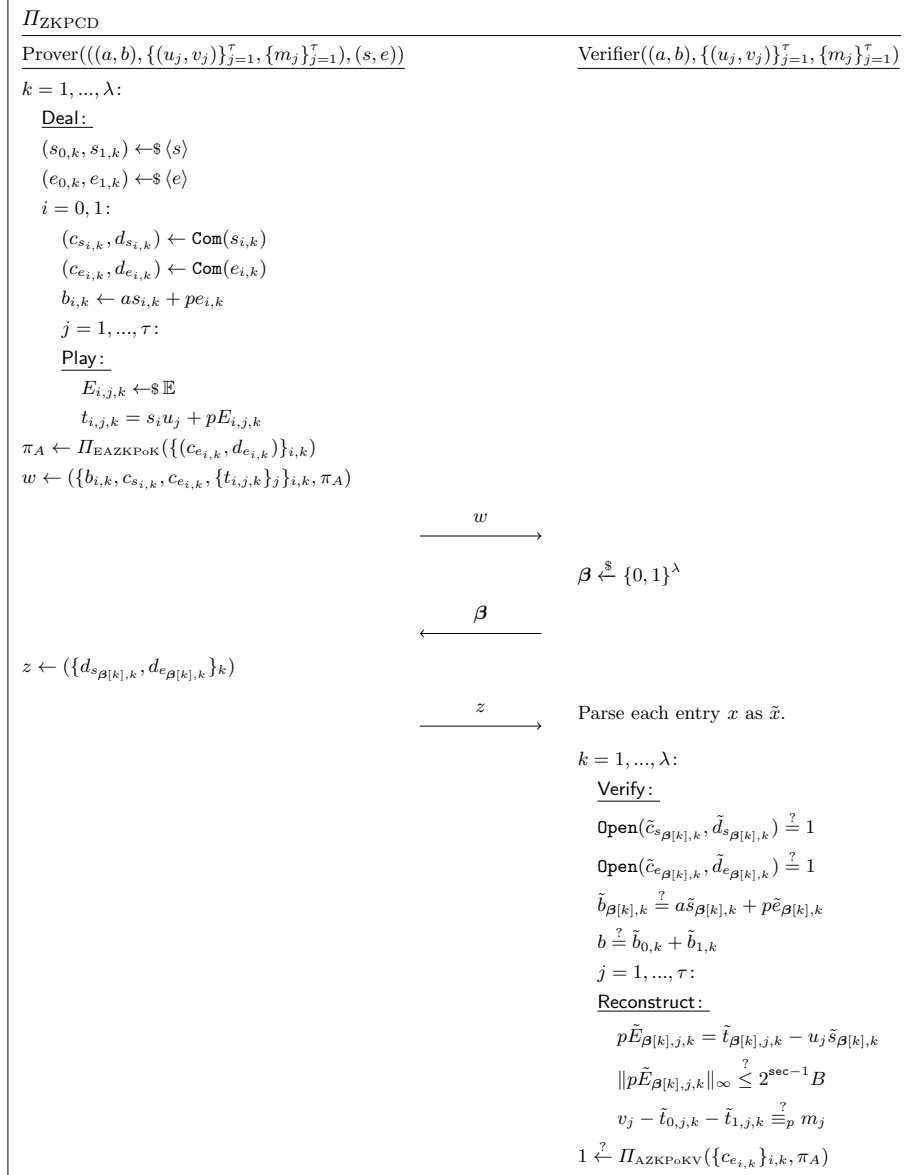


Fig. 6. Zero-knowledge proof of correct decryption.

$\rho_{\beta[k],k}$ of length 2κ bits can be used to deterministically generate the randomness used in $c_{s_{\beta[k],k}}$ and $c_{e_{\beta[k],k}}$, and to generate noise $E_{\beta[k],j,k}$, for all $j = 1, \dots, \tau$.

It follows that, for each $k = 1, \dots, \lambda$, the prover sends either $s_{\beta[k],k}$ or $e_{\beta[k],k}$ and $\rho_{\beta[k],k}$, together with the commitments $c_{s_{1-\beta[k],k}}$ and $c_{e_{1-\beta[k],k}}$, and the partial decryptions $\{t_{1-\beta[k],j,k}\}_{j=1}^{\tau}$. Because we only need a commitment without any special properties to commit to $s_{i,k}$, we let $c_{s_{i,k}} = H(s_{i,k}, r_{s_{i,k}})$. However, since $c_{e_{1-\beta[k],k}}$ is used in the amortized proof of shortness, this commitment is computed using the lattice-based commitment scheme in Section 5.5. We observe that $c_e = c_{e_{1-\beta[k],k}} + a' e_{\beta[k],k}$ is a commitment to $e_{\beta[k],k} + e_{1-\beta[k],k}$, which is supposed to be short. We set σ so that all noise values e are ternary, and use the amortized zero-knowledge protocol directly as described in Section 5.6.

Total communication. The total amount of communication by the prover is

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

The amortized proof is of size $(N + (\lambda + 2)\eta) \log q + 2\kappa(h + \eta \log(l/h))$ bits, however, we might have to run the proof several times to get negligible soundness according to Lemma 1. For a concrete instantiation, we use the example parameters in Table 1, estimated to $\kappa = 100$ bits of security using the LWE-estimator [APS15]. Assuming $\lambda = 100$ we can run the amortized proof two times to get soundness smaller than $2^{-\lambda}$ with the given values for k, l, η , giving a proof size $|\pi_A|$ of less than 86 KB. It follows that for a large number of ciphertexts τ we can ignore the other terms and get a proof size of $\approx 15.87\lambda\tau$ KB.

7.2 Timings

We have not implemented our protocol, but give an estimate for the time it takes to compute our proof of correct decryption based on the elementary operations. The most expensive computations in lattice-based cryptography are sampling elements due to some distribution, and compute multiplications in the ring. We use the NTT to increase the efficiency of the latter. Then we need a fully splitting ring R_q of dimension $N = 2^{11}$, where q is a 62-bit prime of the form $q \equiv 1 \pmod{2N}$, for example $q = 2^{62} - 2^{16} + 1$. Because q is two bits smaller than the word size commonly used on modern computers, modular reductions modulo q are 30 % more efficient than for primes of size 64 bits.

The prover does the following computation in the protocol:

- Sample 2λ uniform elements $s_{0,k}$ and $e_{0,k}$ in R_q . Convert 4λ values to NTT.
- Sample 2λ short vectors r such that $\|r\|_{\infty} = 1$. Convert them to NTT.
- Compute 2λ commitments at the cost of 1 multiplication each in NTT.
- Compute 2λ key-shares b_i at the cost of 1 multiplications each in NTT.
- Convert τ ciphertext-values u_j in R_q to NTT.
- Sample $2\lambda\tau$ uniform but bounded ring elements $E_{i,j,k}$ in R_q .
- Compute $2\lambda\tau$ products $s_i u_j$ in NTT. Convert back from NTT.

Parameter	Explanation	Constraints	Value
N	Dimension	Power of two	2048
q	Ciphertext modulus	$2(1 + 2^{\text{sec}})B_{\text{Dec}} < q \equiv 1 \pmod{2N}$	$\approx 2^{62}$
p	Plaintext modulus		3
κ	Security parameter	Long-term privacy	100 bits
λ	Soundness parameter		10, ..., 100
sec	Statistical security	See [DKL+13, Appendix G]	40
σ	Standard deviation	(ternary error-polynomials)	$\sqrt{2/3}$
B_{∞}, B'_{∞}	Bounds on secrets		1
B_{Dec}	Decryption bound	$\ v - su\ _{\infty} \leq B_{\text{Dec}}$	2^{20}
η	Size of challenge set I	Soundness Lemma 1	50
k	Input size to code	Equal to $N + 7\eta$	2398
l	Length of code	Must be larger than k	5000
h	# Merkle trees	E.g. equal to 2η	100

Table 1. Notation, explanation, constraints and example parameters for the protocol.

Adding everything together, we get total timing:

$$(2\lambda + 2\lambda\tau)\mathsf{U} + (4\lambda + 2\lambda\tau)\mathsf{M} + (6\lambda + (2\lambda + 1)\tau)\mathsf{NTT}.$$

We use the numbers from NFFlib [ABG⁺16] to estimate the timings in our protocol for NTT, multiplication and sampling. Note that the estimates in NFFlib are for $N = 2^{10}$, and the real timings would be slightly larger. The NTT is a bijection and takes $\mathsf{NTT} \approx 14 \mu\text{s}$ to compute each way, while it takes $\mathsf{M} \approx 3 \mu\text{s}$ to multiply two polynomials on NTT-form. It takes $\mathsf{U} \approx 10 \mu\text{s}$ to sample a uniform polynomial in R_q . We assume that scalar multiplications by p and additions comes for free. Sampling ring elements r such that $\|r\|_\infty = 1$ can be done by pure hashing, and we ignore this cost as well. To compensate for the larger N , we assume that the real timings are within a factor 2 of these estimates.

If we ignore the τ -terms, set $\lambda = 100$ and insert the values in Table 1, we get that the overhead of the protocol is less than 25 ms. The prover time for the amortized proof is estimated to be around 2 seconds (see [BLNS20, Table 1]). Hence, the real cost of the protocol is $\approx 2\lambda(\mathsf{U} + \mathsf{M} + \mathsf{NTT})\tau \approx 100\lambda\tau \mu\text{s}$. We note that verification of this proof requires computation of a similar order.

We also note that several parts of our protocol are independent of the ciphertexts being decrypted. A commonly used technique is to move computation to a pre-processing, or offline, phase, where we perform the computations that are independent of the instance we are proving. The Deal-algorithm is independent of the ciphertexts, which means that sampling, key-splitting and computing commitments and zero-knowledge proofs can be done offline.

Assume that τ_{\max} is an upper bound on the number of ciphertexts. Then we can sample $2\lambda\tau_{\max}$ values $E_{i,j,k}$ from \mathbb{E} in advance of running the Play-algorithm. The ciphertexts are input to the decryption protocol, and we can require that the u_j 's are given on NTT form, as this computation is independent of secret information and can be done by e.g. the encryption algorithm. The size of the proof is still the same, but online computation is reduced to $\approx 70\lambda\tau \mu\text{s}$.

8 Comparison to Other Protocols

8.1 Comparison to Π_{DistDec}

We will now sketch an extension of the passively secure distributed decryption protocol Π_{DistDec} given in Section 5.4, which is used in SPDZ [DKL⁺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 Π_{DistDec} -protocol runs as follows:

1. Each party \mathcal{D}_i samples $E_{i,j}$ from \mathbb{E} 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⁺18].
3. Each party \mathcal{D}_i use the amortized ZKP by Baum *et al.* [BBC⁺18] to prove that each $E_{i,j}$ is bounded by $2^{\text{sec}} B/\xi p$, given commitments $\mathbf{c}_{E_{i,j}}$.
4. The verifier checks the relations $(v_j - t_{0,j} - t_{1,j} \bmod q) \equiv m_j \bmod p$ and that all the zero-knowledge proofs are valid.

Ring elements t_j and commitments $\mathbf{c}_{E_{i,j}}$ are of size $N \log q$ and $2N \log q$, respectively. Each proof of linearity $\pi_{L_{i,j}}$ is of size $\approx 9N \log 6\bar{\sigma}$ bits. The amortized proof is of size $4\hat{n}N \log 6\bar{\sigma}$, where $\hat{n} \geq (\kappa + 2)/\log(2N + 1)$. The total size, for each \mathcal{D}_i , is

$$\approx 4\hat{n}N \log(6\bar{\sigma}) + (3N \log q + 9N \log(6\bar{\sigma}))\tau \text{ bits.}$$

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 $\hat{\sigma} \approx 2^{95}\sqrt{\tau}$. However, the amortized proof is not exact, which means that we must increase q to ensure correct decryption. Assuming that $\tau \leq 2^{20}$, we need to increase q by 50 bits. To achieve security $\kappa = 100$ we then need to increase N to 2^{12} . It follows that the proof is of size $\approx 516\tau + 3200$ KB. We conclude that Π_{ZKPCD} is smaller for $\lambda \leq 33$ and approximately $\times 3$ times larger than Π_{DistDec} for $\lambda = 100$.

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 number of repetitions in our construction.

8.2 Comparison to Boschini *et al.* (PQCrypto 2020)

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 somewhat smaller decryption proofs, but that 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 for moderate and large sets of ciphertexts.

8.3 Comparison to Lyubashevsky *et al.* (ePrint 2021)

A recent preprint by Lyubashevsky, Nguyen and Seiler [LNS20] gives a verifiable decryption protocol for the Kyber encapsulation scheme [SAB⁺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 250 is of size 43.6 KB. We note that the message space is smaller than in our protocol, mostly because we are forced to choose larger parameters because of the noise-drowning technique to hide the secret key in the partial decryption shares. We expect the proof for ternary messages, $N = 2^{11}$ and $q \approx 2^{32}$ will be comparable to our proof for $\lambda \approx 60$. They don't provide timings.

8.4 Other Approaches

Recent developments in lattice-based zero-knowledge proofs can be used to prove shortness of vectors as a building block to prove correct decryption. Examples are Beullens [Beu20], Baum and Nof [BN20], Lyubashevsky *et al.* [BBC⁺18, BLS19, ALS20], Yang *et al.* [YAZ⁺19] and Esgin *et al.* [ENS20]. However, proofs of decryption requires both proofs of shortness, range proofs and linear relations, and combining these proofs for comparable parameters as in our scheme will result in proof sizes of several hundred KB or a few MB per ciphertext as in [LNS20]. Also, the running times of these protocols often require the prover to sample Gaussian values and re-run the proof protocol several times for each statement, which increase the timings of the final proof.

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 efficient single-party verifiable decryption protocol for lattice-based cryptography when instantiated with the BGV encryption scheme. It can be made non-interactive using the Fiat-Shamir transform.

The size and efficiency of the protocol is a small factor times $\lambda\tau$, for arbitrary soundness parameter λ and number of ciphertexts τ . The long-term privacy of the protocol κ can be set independently of, and in particular larger than, λ . This allows an interactive instantiation of the protocol to be very efficient, both in size and computation. For $\kappa = 100$ we estimate the proof to be of size $\approx 16\lambda\tau$ KB and the proof/verification time to be only $100\lambda \mu s$ per ciphertext. The prover time can be decreased to only $70\lambda \mu s$ per ciphertext if we allow for preprocessing.

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 amortized proof in the end.

Finally, we compare our solution with other protocols. First, we append zero-knowledge proofs to Π_{DistDec} [DKL⁺13, DPSZ12] and use similar ideas as above to create another one-party verifiable decryption protocol. We compare their efficiency, and conclude that Π_{ZKPCD} beats Π_{DistDec} w.r.t. computational complexity and provides smaller proofs for $\lambda \leq 33$. Secondly, we compare with the protocol by Boschini *et al.* [BCOS20], and observe that even though they provide slightly shorter proofs when λ is larger, the computational complexity is so heavy that the protocol is impractical except for small sets of ciphertexts. Lastly, we admit that the newly published protocol by Lyubashevsky, Nguyen and Seiler [LNS20] provides much smaller proofs than we do for small message-spaces (which we cannot achieve because of constraints inherent to our protocol), while we provide smaller proofs for comparable parameters when $\lambda \leq 60$.

9.2 Future Improvements and Extensions

We consider the following four extensions to our protocol future work.

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.

More size-efficient decryption procedure. The main obstacle in our protocol is the size of q . The modulus is required to be a factor 2^{sec} larger than what is necessary to just ensure correct decryption, to make sure that the decryption protocol does not leak any information about the secret key. This is the only way we know of today to perform verifiable decryption for lattice-based encryption schemes, and any improvement in this area would, most likely, directly improve the efficiency of our scheme as well.

Extension to distributed decryption. We can extend our protocol to more than one party in the following way. We split the private key among n parties, and each of the parties run the proof of correct decryption protocol locally to prove correctness of their partial decryption share. Then we can publicly combine the output to verify the correct decryption.

The size of the proof would increase by a factor n , but the timings would be the same as all the computation would be performed in parallel by each party. Now, the soundness is the same if all n parties are colluding, while the soundness error would decrease if there is at least one honest party involved. This extension

requires a more careful analysis before it can be used in practice. In our dynamic framework the security for the soundness could also be set differently for each player if deemed useful.

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

References

- ABG⁺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.
- 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.
- 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⁺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.
- BCS19. Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*, volume 11959 of *Lecture Notes in Computer Science*, pages 274–302. Springer, Heidelberg, August 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⁺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.
- BGG⁺18. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 565–596. Springer, Heidelberg, August 2018.
- 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.
- BKP13. Rikke Bendlin, Sara Krehbiel, and Chris Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 218–236. Springer, Heidelberg, June 2013.
- BLNS20. Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. More efficient amortization of exact zero-knowledge proofs for LWE. *Cryptology ePrint Archive*, Report 2020/1449, 2020. <https://eprint.iacr.org/2020/1449>.
- BLO18. Carsten Baum, Huang Lin, and Sabine Oechsner. Towards practical lattice-based one-time linkable ring signatures. In David Naccache, Shouhuai Xu, Sihan Qing, Pierangela Samarati, Gregory Blanc, Rongxing Lu, Zonghua Zhang, and Ahmed Meddahi, editors, *ICICS 18: 20th International Conference on Information and Communication Security*, volume 11149 of *Lecture Notes in Computer Science*, pages 303–322. Springer, Heidelberg, October 2018.
- BLS19. Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 176–202. Springer, Heidelberg, August 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.
- Dam10. Ivan Damgård. On Σ -protocols. Private note, 2010. <https://www.cs.au.dk/~ivan/Sigma.pdf>.
- 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.
- DKL⁺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.
- DOTT20. Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n -out-of- n and multi-signatures and trapdoor commitment from lattices. Cryptology ePrint Archive, Report 2020/1110, 2020. <https://eprint.iacr.org/2020/1110>.
- 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.
- ENS20. Muhammed F. Esgin, Ngoc Khanh Nguyen, and Gregor Seiler. Practical exact proofs from lattices: New techniques to exploit fully-splitting rings. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 259–288. Springer, Heidelberg, December 2020.
- 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.

- GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206. ACM Press, May 2008.
- HGT19. Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. In *CCS*, pages 685–702. ACM, 2019.
- 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*, pages 1–8, 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.
- KO21. Stephan Krenn and Michele Orrù. Proposal: Σ -protocols. 4th Annual ZKProof Workshop, 2021. <https://docs.zkproof.org/pages/standards/accepted-workshop4/proposal-sigma.pdf>.
- 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.
- 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.
- LMPR08. Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In Kaisa Nyberg, editor, *Fast Software Encryption – FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 54–72. Springer, Heidelberg, February 2008.
- 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.
- LNS20. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. *Cryptology ePrint Archive*, Report 2020/1448, 2020. <https://eprint.iacr.org/2020/1448>.
- 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. Fucai 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.
- PR06. Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 145–166. Springer, Heidelberg, March 2006.
- SAB⁺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>.
- Scy18. Scytl. Scytl sVote, complete verifiability security proof report - software version 2.1 - document 1.0, 2018. <https://www.post.ch/-/media/post/evoting/dokumente/complete-verifiability-security-proof-report.pdf>.
- SSA⁺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.
- YAZ⁺19. Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, Zuoxia Yu, and William Whyte. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 147–175. Springer, Heidelberg, August 2019.