Gladius: LWR based efficient hybrid public key encryption with distributed decryption

Kelong Cong[1][0000−0002−2636−4406], Daniele Cozzo[1][0000−0001−5289−3769], Varun Maram[2][0000−0002−1607−9062], and Nigel P. Smart[1,3][0000−0003−3567−3304]

[1] imec-COSIC, KU Leuven, Leuven, Belgium.
[2] ETH, Zurich, Switzerland.
[3] University of Bristol, Bristol, UK.
kelong.cong@esat.kuleuven.be,
daniele.cozzo@kuleuven.be,
nigel.smart@kuleuven.be,
vmaram@inf.ethz.ch

Abstract. Standard hybrid encryption schemes based on the KEM-DEM framework are hard to implement efficiently in a distributed manner whilst maintaining the CCA security property of the scheme. This is because the DEM needs to be decrypted under the key encapsulated by the KEM, before the whole ciphertext is declared valid. In this paper we present a new variant of the KEM-DEM framework, closely related to Tag-KEMs, which sidesteps this issue. We then present a post-quantum KEM for this framework based on Learning-with-Rounding, which is designed specifically to have fast distributed decryption. Our combined construction of a hybrid encryption scheme with Learning-with-Rounding based KEM, called Gladius, is closely related to the NIST Round 3 candidate called Saber. Finally, we give a prototype distributed implementation that achieves a decapsulation time of 4.99 seconds for three parties.

1 Introduction

The potential development of quantum computers means that we need to rethink which algorithms are going to be used for public key encryption and signatures; resulting in the subarea called post-quantum cryptography. The early days of post-quantum cryptography looked at how to build basic primitives such as simple public key encryption or signatures. However, now we realise that our existing (pre-quantum) public key algorithms often offer more than what is offered by basic public key primitives. For example one may have group signatures, identity-based encryption, or proofs-of-knowledge of the secret key, etc. In this work, we look at distributed decryption for IND-CCA *hybrid* public key encryption.

Even in the context of pre-quantum cryptography, distributed decryption for hybrid systems is problematic for many schemes, as to maintain security one would need to apply a distributed decryption procedure to the symmetric component, which is rather expensive. This problem, of the difficulty of constructing threshold IND-CCA encryption/encapsulation schemes Π_p, was first pointed out in [LL94] and then elaborated upon in [SG98,SG02]. The problem being that Π_p would seem to require a publicly checkable CCA test. For historical (i.e. impractical) CCA secure public key encryption schemes such as Naor-Yung [NY90] and Dolev-Dwork-Naor [DDN91] the check is simply the verification of a zero-knowledge proof, and is thus publicly verifiable.

However, for almost all practical encryption schemes the check is non-public and thus requires often expensive machinery to deploy in a threshold manner. In [SG98,SG02] Shoup and Gennaro present two schemes (called TDH1 and TDH2) which are IND-CCA and are based on the discrete logarithm problem, for which an efficient threshold decryption algorithm is possible. Both schemes

bear a strong resemblance to Cramer-Shoup encryption [CS98]. These two constructions are however non-hybrid encryption mechanisms, but can be turned into hybrid threshold schemes using the Tag-KEM framework [AGK08].

Our first contribution is to provide two transforms (one secure in the ROM and one secure in the QROM) which supports distributed decryption for hybrid encryption schemes. Our transform is closely related to the previous REACT [OP01] transform, the Tag-KEM framework [AGK08], or the second hybrid-variant of the Fujisaki-Okamoto transform [FO13]. The key take away from our (general) hybrid construction is that the DEM component can be a generic one-time IND-CPA encryption scheme, and the KEM component can be either a *rigid*[4] deterministic OW-CPA secure public key encryption scheme or (with a minor modification) a *rigid* OW-PCA-secure[5] probabilistic scheme. In the case of public-key encryption schemes which are not perfectly correct, i.e. they exhibit decryption errors, we require an additional hardness assumption.

As our second contribution, to utilize our hybrid construction in the post-quantum setting we build a *rigid* deterministic encryption scheme which has a relatively efficient distributed decryption procedure based on the standard (or module) Learning-with-Rounding (LWR) problem. Our scheme is competitive (in terms of execution time and parameters) with Saber, the Learning-with-Rounding based submission in the third round of the NIST competition. Indeed the module-LWR version of our scheme has almost exactly the same parameters as Saber[6], meaning that any run-times for Saber in hybrid encryption mode will be similar to the run-times for our scheme.

Due to the similarity with Saber we name our constructions of a hybrid encryption scheme, which has an efficient distributed decryption operation, based on Learning-with-Rounding, after the Roman sword Gladius; which came in four basic forms: A large one called Gladius–Hispaniensis, a smaller 'standard' one called Gladius–Pompeii, and two related ones called Gladius–Mainz and Gladius–Fulham. In addition, we give a pre-quantum hybrid scheme based on ElGamal encryption and the gap-Diffie–Hellman assumption, along with a methodology to perform a distributed hybrid decryption.

Of the three lattice based finalists in Round 3 of the NIST competition two of them, Crystals-Kyber [SAB+19], and Saber [DKRV19], all construct a hybrid encryption scheme by first building an IND-CPA encryption scheme, and then creating an IND-CCA hybrid scheme using the Fujisaki-Okamoto transform [FO99]. The problem with the Fujisaki-Okamoto design pattern is that the decryption procedure needs to perform a hash to obtain the random coins. In the threshold setting this is a problem as one needs to hash both the DEM key k and the DEM value itself (or the message) in the Fujisaki-Okamoto transform to perform the re-encryption; and this must be done *before* one reveals k and m to the decrypting parties. The hash function used for re-encryption also needs to produce the random values used in encryption, which can be a complicated process to perform in a threshold manner; especially if this involves sampling discrete Gaussians or other distributions which are not 'native' to whichever underlying methodology one is using to perform the threshold decryption.

The other remaining lattice based finalist in Round 3, NTRU [ZCH+19], also builds a traditional KEM, with the difference that the KEM does not require re-encryption. However, NTRU builds

[4] A scheme is defined to be rigid if decryption of a 'ciphertext', which is not the output of an encryption operation, always returns $\perp$.

[5] A scheme is said to be PCA (plain-check attack) secure if it is secure in the presence of an oracle which allows the adversary to check whether a given ciphertext encrypts a given plaintext.

[6] Although there is an issue of having comparable security for these parameters, due to our reliance on LWE in the key generation phase, see later for more details.

a traditional KEM, which requires the DEM to be implemented in a threshold manner so as to maintain the CCA security. Thus threshold variants of all the remaining Round 3 lattice based schemes will be problematic if one wishes to maintain CCA security of the threshold variant.

Of the Round-2 lattice-based systems which did not progress to be finalists in Round-3, FrodoKEM [NAB+19], Round 5 [GZB+19], LAC [LLJ+19], NewHope [PAA+19], and ThreeBears [Ham19], also follow the Fujisaki-Okamoto pattern, bar NTRUprime [BCLv19]. NTRUprime differs from the previous ones in that it is based on a *rigid* deterministic base encryption scheme which is then turned into a KEM using [Den03, Section 6]. However, the underlying rigid deterministic encryption scheme still requires re-encryption to be secure.

1.1 Prior Work and Our Contribution

Threshold Decryption: As stated at the beginning our main goal is to provide an efficient threshold decryption procedure for a post-quantum *hybrid* encryption algorithm. We do this by providing an algorithm which is efficient, within a generic MPC framework, to perform distributed decryption. Thus, on the assumption the algorithm we implement is correct, the security of said algorithm follows from the security of the base MPC framework.

In an earlier work [KLO+19] on distributing the decryption for a Round-1 NIST candidate which was based on Ring-LWE, namely LIMA, a distributed decryption operation was given for a basic (non-hybrid) encryption scheme. An outline for the hybrid scheme was given, but the instantiation would not preserve the CCA security guarantees of the hybrid construction, i.e. the method presented was *not* secure.

From a performance perspective the problem with the distributed decryption of LIMA was that it is a scheme based on the Fujisaki-Okamoto transform. As mentioned above the secure evaluation of the hash function and re-encryption operation is costly in the distributed setting. But this is not the only problem with [KLO+19], the decryption procedure itself is rather complicated in that it requires rounding of integers, for example. In [KLO+19] these two technical complexities meant the protocol (to be fast) was only a 3-party protocol with one dishonest party. The distributed decryption of a single non-hybrid LIMA encryption would take 4.2 seconds, with a similar time for the *insecure* hybrid KEM distributed decapsulation.

Traditionally, in the non-hybrid encryption setting, threshold decryption is preferred using the least amount of interaction, for example see [LY12,SG98,SG02]. Our threshold decryption procedure for our post-quantum hybrid scheme utilizes explicitly generic MPC techniques; thus it definitely does not minimize the level of interaction between the parties needed. An open problem would be to develop a methodology, or scheme, which can utilize the minimal amount of communication possible.

We note that there has been some work on threshold post-quantum signature schemes, e.g. [CS19,CS20,DM20], but the techniques and issues are rather different from those employed and discussed here.

Hybrid Encryption: Hybrid encryption is the standard method to encrypt large message via a public key scheme. The actual message is encrypted via a standard block cipher in a secure AEAD mode, such as AES-GCM. Then the one-time symmetric key for this symmetric encryption scheme is transferred to the recipient using a public key methodology. The traditional method of combining the public key encryption scheme $\Pi_p = (\mathcal{K}_p, \mathcal{E}_p, \mathcal{D}_p)$, with message space $\mathcal{M}_p$, and symmetric key encryption scheme $\Pi_s = (\mathcal{K}_s, \mathcal{E}_s, \mathcal{D}_s)$ into a hybrid scheme $\Pi_h = (\mathcal{K}_h, \mathcal{E}_h, \mathcal{D}_h)$ is called KEM-DEM

[CS03]. Where $\mathcal{K}_\star, \mathcal{E}_\star$ and $\mathcal{D}_\star$ are the various schemes key-generation, encryption and decryption algorithms respectively.

The KEM-DEM method of [CS03] requires Π_s to be a (one-time) IND-CCA symmetric cipher[7] and an IND-CCA KEM scheme Π_p (a KEM is a public key scheme designed to encrypt only symmetric keys). The scheme Π_p encrypts the key k for Π_s, and then Π_s is used to encrypt the message using the key k. In particular the encryption algorithm, outputting (c_1, c_2) for $\mathcal{E}_h$ is along the lines of

$$k \leftarrow \mathcal{M}_p, \quad \mathsf{k} \leftarrow H(k), \quad c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k), \quad c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m).$$

However, there is a problem with this construction when one looks for a distributed variant of the decryption algorithm. Even if the decryption algorithm of the KEM Π_p has an efficient distributed decryption operation one cannot derive an efficient distributed hybrid cipher as the decryption of the scheme Π_s needs to be executed also in a distributed manner. Executing Π_s in a distributed manner for standard symmetric encryption scheme is possible, but very inefficient for long messages.

One obvious way to get around this problem is for the distributed decryption operation for the hybrid cipher Π_h to output k in the clear after the Π_p part has been executed, enabling the decryption using Π_s to be done in the clear. We call such a hybrid scheme 'leaky', as the decryption algorithm leaks the underlying symmetric key even if the symmetric component does not decrypt correctly. This intuitively seems attractive, however it breaks the IND-CCA security of the hybrid scheme Π_h via a trivial attack.

The most popular generic transform to turn a public key encryption scheme into a hybrid scheme in the KEM-DEM paradigm is the Fujisaki-Okamoto transform [FO99,FO13]. This comes in two forms, either (from [FO99])

$$k \leftarrow \mathcal{M}_p, \quad \mathsf{k} \leftarrow H(k), \quad c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k; G(k, m)), \quad c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m),$$

or (from [FO13])

$$k \leftarrow \mathcal{M}_p, \quad \mathsf{k} \leftarrow H(k), \quad c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m), \quad c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k; G(k, c_2)),$$

where G is a hash function which produces the random coins needed by the encryption algorithm $\mathcal{E}_p$. The authors of [FO99,FO13] show that this hybrid scheme, assuming some (mild) technical conditions on the encryption algorithm, is IND-CCA if Π_p is OW-CPA and Π_s is IND-CPA. Note, for the first variant one needs to decyrpt c_2 before one can verify the c_1 component, as the decryption operation $\mathcal{D}_p$ requires re-encryption to perform the necessary CCA checks. Because of this, the first Fujisaki-Okamoto hybrid construction can never be securely "leaky".

The second Fujisaki-Okamoto variant has been proved secure in the quantum random-oracle model in [Zha19], where the scheme Π_p is assumed to be 'well-spread', perfectly correct and OW-CPA secure. This second Fujisaki-Okamoto variant can be considered as a variant of the Tag-KEM framework of [AGK08]. The Tag-KEM framework gives another hybrid construction, which works (roughly speaking in the simplest instance) in the following manner

$$\mathsf{k} \leftarrow \mathcal{K}_s, \quad c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m), \quad c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, \mathsf{k} \| G(c_2))$$

where G is a hash function. This hybrid construction is secure if Π_p is IND-CCA secure and Π_s is one-time IND-CPA secure.

[7] One time meaning that the attacker does not get access to an encryption oracle.

Note in [HHK17] a QROM proof of the Fujisaki-Okamoto transform is given, but this is for the related *non-hybrid* public key scheme given by $c \leftarrow \mathcal{E}_p(\mathsf{pk}, m; G(m))$. However, unlike in [Zha19], the encryption scheme is not assumed to be perfectly correct.

One of the applications of the Tag-KEM framework mentioned in [AGK08] is that of threshold hybrid public key encryption. Their argument is as follows. Since the one-time-pad is one-time IND-CPA secure, outputting m already leaks k. Thus revealing the value k before applying the decryption of c_2 cannot break security, as that would contradict their main theorem. Thus one can apply threshold decryption to obtain the decryption of c_1, leak the key k and then decrypt c_2 in the clear as long as Π_s is the one-time-pad encryption scheme. Unfortunately, the authors of [AGK08] require an IND-CCA secure Π_p.

The authors of [AGK08] provide other constructions requiring weaker properties of Π_p, but each one adds its own complications. Indeed if one thinks of the hash function G, in our construction below, applied to c_1, c_2 and k as a MAC function applied to c_1 and c_2 with key k, then their 'weak KEM+MAC' construction is identical to ours.

In [AT09] a construction of CCA secure Tag-based encryption which has threshold decryption is discussed. Their generic methodology uses one-time signatures and a concrete instantiation is given based on the decisional bilinear Diffie–Hellman assumption in pairing groups. Another construction of a threshold tag-KEM in the Random Oracle model based on the RSA problem is given in [IAHS07].

The solution we propose is to utilize the following modification to the Cramer-Shoup basic construction. Our main construction, which we call Hybrid_1, outputs a ciphertext of the form (c_1, c_2, c_3) where, for a hash function G modelled as a random oracle,

$$k \leftarrow \mathcal{M}_p, \quad \mathsf{k} \leftarrow H(k), \quad c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k), \quad c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m), \quad c_3 \leftarrow G(c_1, c_2, k).$$

The distributed decryption algorithm checks the c_3 component and then 'leaks' the key k in the clear, enabling k to be produced and hence m decrypted from the c_2 component. We show that this scheme is IND-CCA secure, even with this form of leaky decryption, if the scheme Π_p is *rigid*, *deterministic* and OW-CPA, or rigid, randomized and OW-PCA secure, and the scheme Π_s is one-time IND-CPA secure. If the scheme Π_p is not perfectly correct then we require the additional hardness assumption that it is hard for the adversary to construct a message/ciphertext pair (m, c) such that $c = \mathcal{E}_p(\mathsf{pk}, m)$, but $\mathcal{D}_p(\mathsf{sk}, c) = \perp$. We also require in this case that the probability of the encryption scheme having collisions, i.e. two messages which encrypt to the same ciphertext, is negligible when this probability is computed over the space of all possible public/private key pairs.

When Π_p is randomized and OW-PCA, one needs to include c_1 into the hash function G so as to avoid attacks related to re-randomization of the output of $\mathcal{E}_p$. In this latter case, of randomized OW-PCA encryption scheme Π_p, our construction looks most closely related to the REACT transform, from [OP01], which encrypts via

$$k \leftarrow \mathcal{M}_p, \quad \mathsf{k} \leftarrow H(k), \quad c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k), \quad c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m), \quad c_3 \leftarrow G(k, m, c_1, c_2).$$

The authors of [OP01] show that REACT is secure assuming Π_p OW-PCA secure and the scheme Π_s is IND-CPA secure. The REACT transform has a similar problem with the standard KEM-DEM construction above in that it requires c_2 to be decrypted before the check is applied, i.e. m is needed as an input to G.

In the case when Π_p is rigid and deterministic one can drop the component c_1 from the input to G. So our hybrid construction simplifies to

$$k \leftarrow \mathcal{M}_p, \quad \mathsf{k} \leftarrow H(k), \quad c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k), \quad c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m), \quad c_3 \leftarrow G(c_2, k).$$

In this case one can think of our construction as precisely the second Fujisaki-Okamoto construction utilizing the OW-CPA, 'well-spread' public key encryption scheme with encryption algorithm given by

$$\mathcal{E}'_p(\mathsf{pk}, k; r) = (\mathcal{E}_p(\mathsf{pk}, k), r).$$

Thus our construction in this case would be automatically secure in the QROM if one considers only normal decryption oracle queries (i.e. ones which do not leak the key k); assuming that the techniques used in [HHK17] for dealing with non-perfectly correct schemes could be extended to the proof in [Zha19].

However, this hybrid construction seems hard to prove QROM secure when one requires threshold decryption, unless one picks the DEM operation to be a one-time pad encryption scheme. To obtain a full QROM secure efficient hybrid construction with distributed decryption we present a second hybrid construction which adds a ciphertext component, by hashing k with a second hash function H' which has domain and codomain equal to $\mathcal{M}_p$, as well as hashing k via another hash function H'', before passing the result into G; namely we compute

$$k \leftarrow \mathcal{M}_p, \ \mathsf{k} \leftarrow H(k), \ \mu \leftarrow H'(k),$$
$$c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k), \ c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m), \ c_3 \leftarrow G(c_2, \mu), \ c_4 \leftarrow H''(k).$$

This construction, which we call Hybrid_2, is proved secure, in the QROM, using the techniques of [TU16].

Most of our technical difficulties arise from the fact we want both efficient distributed decryption and an efficient DEM operation. If we take an AES-based DEM then the output of the hash function H will be a bit vector in $\{0,1\}^{|\mathsf{k}|}$. But the input k will be 'native' to the underlying public key scheme, and thus in general an element of a set such as $\mathbb{F}_p^n$, for some modulus p. This means H needs to map from one arithmetic domain to another. It is to avoid needing to do this in a secure way during distributed decryption that we 'leak' the key k and not the key k. This problem does not occur with the hash function G as we are free to select the hash function so that it can be evaluated securely. In our QROM construction using the c_4 component we need to evaluate H' and H'' securely before releasing k, but this can be done as efficiently as evaluating G, by selecting the hash functions H' and H'' in an appropriate way.

Learning-with-Rounding: After detailing our main hybrid constructions we go on to discuss how one can instantiate a suitable KEM in the post-quantum setting. For this we utilize the Learning-With-Rounding (LWR) based deterministic encryption algorithm first presented in [XXZ12], and then refined in [AKPW13]. This is itself inspired by the trapdoor LWE key generation procedure introduced by Micciancio and Peikert [MP12]. We present an explicit construction, including suggested parameters sizes, and compare the resulting scheme with current NIST PQ-candidates such as Saber [DKRV19]. Our basic construction utilizes the fact that LWR encryption is deterministic in nature.

To prove our main hybrid constructions secure we need to assume a new hard problem, which we dub the Large-Vector-Problem (LVP) problem. Informally, this problem says that for a given LWE key $(A, A \cdot R_1 + R_2)$ with $A \in \mathbb{F}_q^{n \times n}$ uniformly randomly chosen and $R_1, R_2 \in \mathbb{F}_q^{n \times sn}$ but with 'small' entries, it is hard to find a small vector $\mathbf{m}$ such that $R_1 \cdot \mathbf{m}$ is 'relatively big'.

The Gladius Family of Hybrid Ciphers: Combining our LWR-based rigid deterministic OW-PCA encryption scheme with our hybrid constructions we obtain a post-quantum secure hybrid cipher, which supports efficient distributed decryption. We can actually derive many variants depending on the choice of Hybrid_1 or Hybrid_2, the choice of the DEM, and the choice of using plain LWR or Module-LWR. We focus on four specific variants of this construction; Gladius–Hispaniensis (based on Hybrid_1 and plain LWR), Gladius–Pompeii and Gladius–Mainz (based on Hybrid_1 and Module-LWR), and and Gladius–Fulham (based on Hybrid_2 and Module-LWR).

Gladius–Hispaniensis, Gladius–Pompeii and Gladius–Fulham all assume *any* one-time IND-CPA secure DEM. For Gladius–Hispaniensis and Gladius–Pompeii we obtain (expected) security in the QROM when the scheme is considered as a standard hybrid encryption scheme, and security in the ROM when we consider the scheme in the threshold setting (due to the additional leakage required). The expected QROM security, which we denote by QROM*, comes from the fact that Zhandry's proof [Zha19], for the second Fujisaki–Okamoto transform, only applies to perfectly correct schemes. We also present a third variant Gladius–Mainz which provides QROM* security in the threshold setting. but this requires the DEM to be a one-time-pad (OTP), and requires a more expensive distributed decryption algorithm. Our fourth variant, Gladius–Fulham, utilizes the second hybrid transform mentioned above, but can achieve full QROM security (including for non-perfectly correct schemes Π_p) even when one allows the leakage from the decryption oracle required in a distributed decryption operation.

In summary the properties of our four schemes are given by the following table, where ✓* in the QROM column refers to the above QROM* caveat. We also note in the table how many secure hash function operations need to be executed by the distributed decryption algorithm.

	Hard Problem	Hybrid	DEM	Standard QROM Secure	Threshold QROM Secure	Threshold ROM Secure	No. Secure Hashes
Gladius–Hispaniensis	LWR	1	Generic	✓*	✗	✓	1
Gladius–Pompeii	Module-LWR	1	Generic	✓*	✗	✓	1
Gladius–Mainz	Module-LWR	1	OTP	✓*	✓*	✓	1
Gladius–Fulham	Module-LWR	2	Generic	✓	✓	✓	3

2 Preliminaries

Notation: By way of notation we let $a \leftarrow A$ denote randomly assigning a value a from a set A, where we assume a uniform distribution on A. If A is an algorithm, we let $a \leftarrow A$ denote assignment of the output, where the probability distribution is over the random coins of A; we also let $a \leftarrow b$ be a shorthand for $a \leftarrow \{b\}$, i.e. to denote normal variable assignment. All functions, distributions on single values are extended to vectors and matrices component-wise without mention.

When performing reduction modulo p (or for any other modulus) this is performed so that the result is in the centred interval, i.e. $(-p/2, \ldots, p/2]$, unless explicitly stated otherwise. Vectors will be column vectors. We let $\|\mathbf{x}\|_\infty$ denote the max-norm on a vector $\mathbf{x}$.

Probabilities and Norms: All our ciphertexts are deterministic encryptions of a message. Thus to work out decryption failure probability we need to measure this over the space of all messages; as there are no other random coins. We want the probability of a decryption failure to be bounded by $2^{-\epsilon}$. From ϵ we define $\mathfrak{c}$ such that $\mathsf{erfc}(\mathfrak{c}) \approx 2^{-\epsilon}$. For example, if $\epsilon = 128$ then we define $\mathfrak{c} = 9.3$.

Let X_i denote a set of random variables, for $i = 1, \ldots, t$, each with variance V_i and each with mean zero. The variance of the random variable obtained by taking the product of t values x_i, where x_i is sampled from X_i, is given by $V_t = \prod V_i$.

If we then take a sum of n such products, then by the Central Limit Theorem, the resulting sum will, for large n, behave as a normal distribution with mean zero and variance $n \cdot V_t$. This means that with probability $1 - 2^{-\epsilon}$ the value of such a sum of n such products will be less than $\mathfrak{c} \cdot \sqrt{n \cdot V_t}$. Note, one could replace the constant $\mathfrak{c}$ here by a constant $\mathfrak{c}_t$ defined such that $\mathsf{erfc}(\mathfrak{c}_t)^t \approx 2^{-\epsilon}$, but $\mathfrak{c}_t \leq \mathfrak{c}$ for all t.

If we let $M \in \mathbb{Z}^{n \times n}$ denote a matrix whose entries have been sampled from $\mathcal{D}_\sigma$, then with probability $1 - 2^{-\epsilon}$ each entry of M is bounded by $\mathfrak{c} \cdot \sigma$, and so with very high probability we have that $\|M\|_\infty \leq \mathfrak{c} \cdot n \cdot \sigma$. But using this directly to measure decryption failures results in far too pessimistic bounds, thus instead we aim to bound $\|\mathbf{x} \cdot M\|_\infty$ for different distributions of the vector $\mathbf{x}$ in our analysis.

Let $\mathbf{x}$ denote a vector sampled in $\mathbb{Z}^n$ from a distribution with mean zero and variance V. Then the entries of $\mathbf{x}^\mathsf{T} \cdot M$ are a sampled from a distribution which is the sum of n variables, each of which is a product of a quantity sampled from a distribution of variance V and a quantity sampled from a distribution of variance σ^2. Thus the entries of $\mathbf{x}^\mathsf{T} \cdot M$ act like they come from a normal distribution of mean zero and variance $n \cdot V \cdot \sigma^2$. Thus in particular we have with probability at least $1 - 2^{-\epsilon}$ that each entry of $\mathbf{x}^\mathsf{T} \cdot M$ is bounded by $\mathfrak{c}_2 \cdot \sqrt{n \cdot V} \cdot \sigma$. So with probability approximately $1 - n \cdot 2^{-\epsilon}$ we have $\|\mathbf{x}^\mathsf{T} \cdot M\|_\infty \leq \mathfrak{c}_2 \cdot \sqrt{n \cdot V} \cdot \sigma \leq \mathfrak{c} \cdot \sqrt{n \cdot V} \cdot \sigma$.

Learning-with-Errors and Learning-with-Rounding: We let σ denote a standard deviation, and we let $\mathcal{D}_\sigma$ denote a distribution which 'looks like' a discrete Gaussian distribution with standard deviation σ. In practice this can be generated by the NewHope methodology [ADPS16], namely if we have $\sigma = \sqrt{(B+1)/2}$ then we sample from $\mathcal{D}_\sigma$ by generating $2 \cdot B + 2$ random bits (b_i, b'_i) for $i = 0, \ldots, B$, and then generating a sample by computing $\sum_{i=0}^{B}(b_i - b'_i)$.

Given a secret vector $\mathbf{s} \in \mathbb{F}_q^n$, then a Learning-with-Errors (LWE) sample is a pair $(A, A \cdot \mathbf{s} + \mathbf{e})$ where $A \in \mathbb{F}_q^{m \times n}$ is chosen uniformly at random and $\mathbf{e} \leftarrow \mathcal{D}_\sigma$. The decision LWE problem is to distinguish LWE samples from uniformly random samples $(A, \mathbf{u})$, for $\mathbf{u} \leftarrow \mathbb{F}_q^m$, we denote this problem by $\mathsf{LWE}_{q,(m,n),\sigma}$. The search LWE problem is to recover the secret vector $\mathbf{s}$ from a set of LWE samples. For suitable choices of the parameters both these problems are known to be equivalent and assumed to be hard. Suitable parameters to ensure hardness given known attack algorithms can be found using Albrecht's *LWE-estimator* tool[8].

For integers p and q we define the following map

$$\lfloor x \rceil_p : \begin{cases} \mathbb{Z}_q \longrightarrow & \mathbb{Z}_p \\ x \longmapsto & \lceil x \cdot p/q \rfloor \pmod p \end{cases}$$

where $\lfloor \cdot \rceil$ is the round to nearest integer function, with rounding towards zero in the case of values of the form $i/2$ for i an odd integer. If the input value $x \in (-q/2, \ldots, q/2]$ then the final reduction modulo p is only required (if p does not divide q) when the rounding ends up being outside the interval $(-p/2, \ldots, p/2]$, which happens with probability about $1/p$, resulting in needing a single addition of p to accomplish the reduction modulo p.

Given a secret vector $\mathbf{s} \in \mathbb{F}_q^n$, then a Learning-with-Rounding (LWR) sample is a pair $(A, \lfloor A \cdot \mathbf{s} \rceil_p)$ where $A \in \mathbb{F}_q^{m \times n}$ is chosen uniformly at random. The decision LWR problem is to distinguish LWR samples from uniformly random samples $(A, \mathbf{u})$, for $\mathbf{u} \leftarrow \mathbb{F}_p^n$, we denote this problem by $\mathsf{LWR}_{q,p,(m,n)}$. The search problem is similarly defined as the problem of recovering $\mathbf{s}$ from a number of LWR samples.

[8] `https://bitbucket.org/malb/lwe-estimator/src/master/`

Module Learning-with-Errors and Learning-with-Rounding: The extension of the plain-LWE and LWR settings to module variants is standard. Here we define the notation we shall use. We will utilize the 2-power cyclotomic ring $\mathcal{R} = \mathbb{Z}[X]/\Phi_{2\cdot n}(X)$ for $n = 2^N$, hence $\Phi_{2n}(X) = X^n + 1$. For an integer $q \geq 2$ we then obtain the ring $\mathcal{R}_q$ by setting $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. We let $\mathcal{D}_\sigma(\mathcal{R}_q)$ denote the distribution obtained by sampling an element from $\mathcal{R}_q$ with coefficients drawn from $\mathcal{D}_\sigma$.

As a norm on elements in $\mathcal{R}$ we will use the infinity norm of the polynomial embedding, namely for $\mathbf{a} \in \mathcal{R}$ we write $\mathbf{a} = \sum_{i=0}^{n-1} a_i \cdot X^i$ then we set $\|\mathbf{a}\|_\infty := \max_{i=1,\ldots,l} \|a_i\|_\infty$. It is well known that for this norm, and this ring, we have for $a, b \in \mathcal{R}$ that $\|a \cdot b\|_\infty \leq n \cdot \|a\|_\infty \cdot \|b\|_\infty$. However, just as before if we consider the case when the coefficients of a are samples according to a distribution with zero mean and variance V, and the coefficients of $\mathbf{b}$ are samples according to the distribution $\mathcal{D}_\sigma(\mathcal{R}_q)$ then with probability $1 - n \cdot 2^{-\epsilon}$ we have $\|a \cdot b\|_\infty \leq \mathfrak{c} \cdot \sqrt{n \cdot V} \cdot \sigma$. To see this, simply look at the matrix representation of the ring. In a similar manner we have that if $\mathbf{a}, \mathbf{b} \in \mathcal{R}^d$ with the coefficients of the entries of $\mathbf{a}$ sampled according to a distribution with zero mean and variance V, and the vector $\mathbf{b}$ is sampled according to $\mathcal{D}_\sigma(\mathcal{R}_q)^d$ then with probability $1 - n \cdot d \cdot 2^{-\epsilon}$ we have $\|a \cdot b\|_\infty \leq \mathfrak{c} \cdot \sqrt{n \cdot d \cdot V} \cdot \sigma$.

Given the ring $\mathcal{R}_q$ one can define the $\mathcal{R}_q$-module $\mathcal{R}_q^d$, for any integer $d \geq 1$ in the standard manner. A module variant of the decision LWE problem can be given as follows:

Definition 2.1 (Decision Module-LWE Problem (D-MLWE)). *The D-MLWE problem is parametrized by integers $q, d, n, m \geq 1$ and a standard deviation σ. Given a fixed secret vector $\mathbf{s} \leftarrow \mathcal{R}_q^d$, a random $\mathbf{a} \leftarrow \mathcal{R}_q^d$ and m samples of either the form $(\mathbf{a}, \mathbf{a}\cdot\mathbf{s}+\mathbf{e}) \in \mathcal{R}_q^d \times \mathcal{R}_p$ or of the form $(\mathbf{a}, \mathbf{u}) \in \mathcal{R}_q^d \times \mathcal{R}_p$, where $\mathbf{u}$ is a uniformly random element in $\mathcal{R}_q$ and $\mathbf{e}$ is drawn from $\mathcal{D}_\sigma(\mathcal{R}_q)$, the problem is to decide which is the case with non-negligible advantage.*

We denote this problem by $\mathsf{Module\text{-}LWE}_{q,(m,n,d),\sigma}$, again when $d = 1$ we obtain the decision-Ring-LWE problem $\mathsf{Ring\text{-}LWE}_{q,(m,n),\sigma}$, and when $n = 1$ we obtain the plain-LWE problem in dimension d, i.e. $\mathsf{Module\text{-}LWE}_{q,(m,1,d),\sigma} = \mathsf{LWE}_{q,(m,d),\sigma}$. The equivalent search problem can be defined in the normal way.

The LWR function $\lfloor x \rceil_p$ can be extended to the polynomial embedding of $\mathcal{R}$ and thus we obtain a map $\lfloor \mathbf{x} \rceil_p : \mathcal{R}_q \longrightarrow \mathcal{R}_p$. From this we can derive the following module variant of the decision LWR problem:

Definition 2.2 (Decision Module-LWR Problem (D-MLWR)). *The D-MLWR problem is parametrized by integers $q, p, d, n, m \geq 1$. Given a fixed secret vector $\mathbf{s} \leftarrow \mathcal{R}_q^d$, a random $\mathbf{a} \leftarrow \mathcal{R}_q^d$ and m samples of either the form $(\mathbf{a}, \lfloor \mathbf{a} \cdot \mathbf{s} \rceil_p) \in \mathcal{R}_q^d \times \mathcal{R}_p$ or of the form $(\mathbf{a}, \lfloor \mathbf{u} \rceil_p) \in \mathcal{R}_q^d \times \mathcal{R}_p$, where $\mathbf{u}$ is a uniformly random element in $\mathcal{R}_q$, the problem is to decide which is the case with non-negligible advantage.*

We denote this problem by $\mathsf{Module\text{-}LWR}_{q,p,(m,n,d)}$, when $d = 1$ we obtain the decision-Ring-LWR problem $\mathsf{Ring\text{-}LWR}_{q,p,(m,n)}$, and when $n = 1$ we obtain the plain-LWR problem in dimension d, i.e. $\mathsf{Module\text{-}LWR}_{q,p,(m,1,d)} = \mathsf{LWR}_{q,p,(m,d)}$. The equivalent search problem can be defined in the normal way.

Relation between (Module-) LWE and (Module-) LWR: The search (Module-) LWE and (Module-) LWR problems are linked theoretically by the following theorem [BGM+16, Theorem 1 and 2][9].

[9] The result in [BGM+16] is only given for normal and Ring LWE/LWR, but extending the result to the module variants is immediate.

Theorem 2.1. *Let p, q, n, d, m and B be integers such that $q > 2 \cdot p \cdot B$. For every algorithm* Learn' *there is an algorithm* Learn *such that*

$$\begin{aligned}\Pr_{A,\mathbf{s},\mathbf{e}}[\mathsf{Learn}'(A, A \cdot \mathbf{s} + \mathbf{e}) = \mathbf{s}] &\geq \Pr_{A,\mathbf{s},\mathbf{e}}[\mathsf{Learn}(A, \lfloor A \cdot \mathbf{s} + \mathbf{e} \rceil_p) = \mathbf{s}] \\ &\geq \frac{\Pr_{A,\mathbf{s}}[\mathsf{Learn}(A, \lfloor A \cdot \mathbf{s} \rceil_p) = \mathbf{s}]^2}{(1 + 2 \cdot p \cdot B/q)^{n \cdot m \cdot d}}\end{aligned}$$

where $A \leftarrow \mathcal{R}_q^{m \times d}$, the noise $\mathbf{e}$ is independent over all m coordinates, B-bounded and balanced in each coordinate, and $\mathbf{s} = (s_i) \in \mathcal{R}_q^d$ is chosen with any distribution supported such that $s_i \in \mathcal{R}_q^*$ for some i.

Note, the first inequality is not from [BGM+16] but it is immediate. To apply this result, we would take $B = \mathfrak{c} \cdot \sigma$, for some suitable constant $\mathfrak{c}$.

The fact that square of the LWR advantage is bounded by the LWE advantage implies that one will need larger parameters to bound the LWR advantage by a given value, than to bound the LWE advantage by the same value. Thus using this theoretical reduction will result in very large parameters indeed. To avoid the problem with the above reduction submissions to the NIST Post-Quantum cryptography competition based on LWR, such as Saber [DKRV18,DKRV19], estimate their parameters by using the best attack scenario. In other words the security is estimated using Albrecht's LWE-estimator directly, or by assuming the above theorem is an exact inequality between the various one-way advantages.

This approach is examined in detail in [ACD+18], where to utilize Albrecht's tool the authors need to translate the LWR parameters into LWE parameters. In [ACD+18] this is done by setting the LWE standard deviation to be

$$\sigma = \sqrt{\frac{(q/p)^2 - 1}{12}}.$$

The Large Vector Problem: We also need to give a new hardness assumption, which we call LVP. Consider the following experiment. The challenger constructs a matrix $A_1 \in \mathbb{F}_q^{n \times n}$ uniformly at random, and then selects $R_1, R_2 \in \mathbb{F}_q^{n \times n}$ with entries selected from the distribution $\mathcal{D}_\sigma$. The challenger constructs $A_2 = A_1 \cdot R_1 + R_2$ and gives the pair (A_1, A_2) to the adversary $\mathcal{A}$. The adversary's goal is to come up with a vector $\mathbf{x} \in [-1/2, \ldots, 1/2]^n$ such that

$$\|R_1 \cdot \mathbf{x}\|_\infty \geq \mathfrak{c} \cdot \sigma \cdot \sqrt{n}/2$$

for some constant $\mathfrak{c}$.

We note that the probability that there are *no solutions at all* to the above problem (when we sample over all keys) is $1 - \mathsf{erfc}(\mathfrak{c})$. The probability that there are **ANY** solutions to this problem is already very small if $\mathfrak{c}$ is large enough. Thus for randomly chosen R_1 and $\mathfrak{c}$ large enough, the adversary already has an impossible task (i.e. information theoretically impossible) in solving LVP.

We also note that if one can solve the search-LWE problem for the pair (A_1, A_2) then finding such a $\mathbf{m}$ is potentially trivial (if such a $\mathbf{m}$ exists). In the 'unlucky' event that there is a solution, since R_1 is hidden (due to search-LWE being hard), the adversary is left with outputting a small vector and 'hoping' it works. Sampling over all keys and messages we have with probability $\mathsf{erfc}(\mathfrak{c}')^2$ that

$$\|R_1 \cdot \mathbf{m}\|_\infty \geq \mathfrak{c}' \cdot \sigma \cdot \sqrt{n/12}.$$

The right-hand-side here will be bigger than our desired bound of $\mathfrak{c} \cdot \sigma \cdot \sqrt{n}/2$ when $\mathfrak{c}' > \mathfrak{c} \cdot \sqrt{12}/2$. Thus the probability of a random vector satisfying the bound is $\mathsf{erfc}(\mathfrak{c} \cdot \sqrt{12}/2)^2$. This will be less than 2^{-128} when $\mathfrak{c} > 3.8$ and will be less than 2^{-256} when $\mathfrak{c} > 5.4$.

The adversary can obviously select a different small set than we suggest, but this would simply changes the bound from $\sqrt{n/12} * \sigma$ to $\mathfrak{c}' * \sqrt{n} * \sigma$ for another constant $\mathfrak{c}'$. The final effect is marginal, since by increasing $\mathfrak{c}$ we *could* remove this possible, but highly unlikely, attack entirely. Thus selecting a $\mathfrak{c}$ which does not eliminate this attack is purely an "optimization" to squeeze a little extra from the concrete parameters. Concretely it allows us to select a smaller $\mathfrak{c}$, in particular 3.8 instead of 9.3, or 5.4 instead of 13.2. This equates to similar size saving in the final parameters.

Thus we define the advantage of an adversary $\mathcal{A}$ against this hard problem as

$$\mathsf{Adv}^{\mathsf{LVP}}_{\mathcal{A}}(n, \mathfrak{c}, \sigma) = \Pr\Big[A_1 \leftarrow \mathbb{F}_q^{n \times n},\ R_1, R_2 \leftarrow \mathcal{D}_\sigma^{n \times n},\ A_2 = A_1 \cdot R_1 + R_2, \\ \mathbf{m} \leftarrow \mathcal{A}(A_1, A_2) \quad : \quad \|R_1 \cdot \mathbf{m}\|_\infty \geq \mathfrak{c} \cdot \sigma \cdot \sqrt{n}/2\Big].$$

Asymmetric and Symmetric Encryption: An asymmetric encryption scheme is a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, all of which are probabilistic polynomial time (PPT) algorithms. We let $\mathcal{M}$ denote the plaintext space of Π, $\mathcal{C}$ the ciphertext space and $\mathcal{R}$ the space of random coins of Π. The key generation algorithm $\mathcal{K}$ takes as input 1^t, where t is a security parameter and outputs a public/private key pair $(\mathsf{pk}, \mathsf{sk})$. A symmetric encryption scheme is one in which $\mathsf{pk} = \mathsf{sk}$.

The algorithm $\mathcal{E}(\mathsf{pk}, m; r)$ takes a message $m \leftarrow \mathcal{M}$, a public key pk and random coins $r \leftarrow \mathcal{R}$ and returns a ciphertext c. The decryption algorithm $\mathcal{D}(\mathsf{sk}, c)$ recovers the message m or returns the special symbol $\perp$. For correctness we require

$$\Pr\Big[\ \mathcal{D}(\mathsf{sk}, c) = m \ : \ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^t),\ m \leftarrow \mathcal{M},\ r \leftarrow \mathcal{R},\ c \leftarrow \mathcal{E}(\mathsf{pk}, m; r)\ \Big] \\ = 1 - \delta,$$

where δ is an exponentially small probability of decryption failure. If $\delta = 0$ we say the scheme is perfectly correct. A public key scheme will be called *deterministic* if $\mathcal{R}$ contains only the empty string (or equivalently one element), otherwise it will be called *randomized*.

A scheme which is not perfectly correct can exhibit two forms of decryption failures; either two messages could map under encryption to the same ciphertext or a valid ciphertext could decrypt to $\perp$. For the first case we say an encryption scheme is δ_c-Collision Free if

$$\Pr_{(\mathsf{sk},\mathsf{pk}) \leftarrow \mathcal{K}(1^t)}\Big[\ \exists\ m_1, m_2 \in \mathcal{M},\ \exists\ r_1, r_2 \in \mathcal{R} \quad : \\ m_1 \neq m_2,\ \ \mathcal{E}(\mathsf{pk}, m_1; r_1) = \mathcal{E}(\mathsf{pk}, m_2; r_2)\ \Big] = \delta_c.$$

A perfectly correct encryption scheme is 0-Collision Free.

For the second case of decryption failure we consider the following game, which we call $\perp$-Aware. The adversary $\mathcal{A}$ is given the public key pk and is required to come up with a plaintext/ciphertext pair (m, c) such that $c = \mathcal{E}(\mathsf{pk}, m)$ but $\mathcal{D}(\mathsf{sk}, c) = \perp$. We define

$$\mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi,\mathcal{A}}(t) = \Pr\Big[\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^t),\ (m, c) \leftarrow \mathcal{A}(\mathsf{pk}) \quad : \\ c = \mathcal{E}(\mathsf{pk}, m),\ \ \mathcal{D}(\mathsf{sk}, c) = \perp\ \Big]$$

and say that Π is $\perp$-Aware if $\mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi,\mathcal{A}}(t)$ is a negligible function of t for all PPT $\mathcal{A}$. Note, if Π is perfectly correct then $\mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi,\mathcal{A}}(t) = 0$.

An asymmetric encryption scheme is said to be *rigid*, see [BP18] (where the definition is given just for deterministic schemes, but the generalization to probabilistic schemes is immediate) if

$$\Pr\left[\ (\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{K}(1^t),\ c \leftarrow \mathcal{C} \setminus \mathcal{C}^{\perp},\ \exists r \in \mathcal{R},\ \ :\ \mathcal{E}(\mathsf{pk}, \mathcal{D}(\mathsf{sk}, c); r) = c\ \right] = 1,$$

where $\mathcal{C}^{\perp} \subset \mathcal{C}$ is the set of all ciphertexts $c \in C$ for which $\mathcal{D}(\mathsf{sk}, c) = \perp$. The effect of rigidity is that unless c is the output of $\mathcal{E}(\mathsf{pk}, m; r)$ for some m and r, then decryption will always return $\perp$. ElGamal is an example of a perfectly correct, rigid probabilistic scheme as every ciphertext pair $(c_1 = g^r, c_2 = m \cdot h^r)$ corresponds to the encryption of some message.

If we let $\|X\|$ be the infinity norm on the probability space X of a finite set S, then the min-entropy of X is $-\log \|X\|$. A randomized asymmetric encryption scheme is said to be γ-spread if

$$\max_{y \in \{0,1\}^*} \Pr\left[r \leftarrow \mathcal{R} : y = \mathcal{E}(\mathsf{pk}, m; r)\right] \geq \gamma$$

for all $(\mathsf{pk}, \mathsf{sk})$ output by $\mathcal{K}$ and all $m \in \mathcal{M}$. A scheme is said to be *well-spread* if $\gamma = \omega(\log t)$. This basically means that the probability of a specific ciphertext occurring is negligibly small.

Note, if the set $\mathcal{R}$ is suitably large then we can turn a deterministic scheme Π_p into a randomized well-spread scheme Π'_p by setting $\mathcal{E}'_p(\mathsf{pk}, k; r) = (\mathcal{E}_p(\mathsf{pk}, k), r)$. It is from this observation, the QROM* security in the standard hybrid (non-leaky) encryption model for our construction based on deterministic public key encryption, mentioned in the introduction, follows.

IND-CPA, IND-CCA and IND-PCA Security: Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ denote a randomized encryption scheme (either symmetric or asymmetric) as above, and let $\mathcal{A}$ denote an adversary. We associate to $\Pi, \mathcal{A}$ the following experiments.

Experiment $\mathsf{Exp}^{\mathsf{ind}-\mathsf{atk}-0}_{\Pi,\mathcal{A}}(t)$	Experiment $\mathsf{Exp}^{\mathsf{ind}-\mathsf{atk}-1}_{\Pi,\mathcal{A}}(t)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{K}(1^t)$	$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{K}(1^t)$
$(m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{pk})$	$(m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{pk})$
$r \leftarrow \mathcal{R}$	$r \leftarrow \mathcal{R}$
$c^* \leftarrow \mathcal{E}(\mathsf{pk}, m_0; r)$	$c^* \leftarrow \mathcal{E}(\mathsf{pk}, m_1; r)$
$b \leftarrow \mathcal{A}_2(c^*, \mathsf{state})$	$b \leftarrow \mathcal{A}_2(c^*, \mathsf{state})$
If $b = 0$ then return one	If $b = 0$ then return one
Else return zero.	Else return zero.

We require that the adversary's messages m_0 and m_1 have the same length length, so as to avoid trivial distinguishing attacks. This setup defines three games. The first is where $\mathsf{atk} = \mathsf{cpa}$ in which the adversary $\mathcal{A}$ is given no additional helper oracles. Note, in the symmetric case the above security notion is that of one-time encryption, i.e. the adversary only ever sees the encryption of a single message under the key[10]. The second is where $\mathsf{atk} = \mathsf{cca}$ in which the adversary is given a decryption oracle which on input of $c \neq c^*$ will return $\mathcal{D}(\mathsf{sk}, c)$ and the third is where $\mathsf{atk} = \mathsf{pca}$ in which the adversary is given a so-called plaintext checking oracle which on input of (m, c) checks whether c is

[10] As we are dealing with one-time security one should in the symmetric case call this $\mathsf{atk} = \mathsf{pass}$, i.e. passive security. But to maintain consistency with much of the literature we call this one-time IND-CPA security.

a valid encryption of m. We define the *advantage* of the adversary $\mathcal{A}$ against Π in these games as

$$\mathsf{Adv}^{\mathsf{ind-atk}}_{\Pi,\mathcal{A}}(t) = \left|\Pr\left[\ \mathsf{Exp}^{\mathsf{ind-atk}-0}_{\Pi,\mathcal{A}}(t) = 1\ \right] - \Pr\left[\ \mathsf{Exp}^{\mathsf{ind-atk}-1}_{\Pi,\mathcal{A}}(t) = 1\ \right]\right|.$$

We say that the scheme Π is IND-CPA (resp. IND-CCA or IND-PCA) secure if $\mathsf{Adv}^{\mathsf{ind-cpa}}_{\Pi,\mathcal{A}}(t)$ (resp. $\mathsf{Adv}^{\mathsf{ind-cca}}_{\Pi,\mathcal{A}}(t)$ or $\mathsf{Adv}^{\mathsf{ind-pca}}_{\Pi,\mathcal{A}}(t)$) is negligible for every PPT $\mathcal{A}$.

OW-CPA, OW-CCA and OW-PCA Security: Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ denote a randomized encryption scheme (either symmetric or asymmetric) as above, and let $\mathcal{A}$ denote an adversary. We associate to $\Pi, \mathcal{A}$ the following experiment

Experiment $\mathsf{Exp}^{\mathsf{ow-atk}-0}_{\Pi,\mathcal{A}}(t)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^t)$
$m \leftarrow \mathcal{M}$
$r \leftarrow \mathcal{R}$
$c^* \leftarrow \mathcal{E}(\mathsf{pk}, m_0; r)$
$m' \leftarrow \mathcal{A}(\mathsf{pk}, c^*)$
If $m = m'$ then return one
Else return zero.

Again this setup defines three games; one where $\mathsf{atk} = \mathsf{cpa}$ in which the adversary $\mathcal{A}$ is given no additional helper oracles, one where $\mathsf{atk} = \mathsf{cca}$ in which the adversary is given a decryption oracle which on input of $c \neq c^*$ will return $\mathcal{D}(\mathsf{sk}, c)$ and one where $\mathsf{atk} = \mathsf{pca}$ in which the adversary is given a so-called plaintext checking oracle which on input of (m, c) checks whether c is a valid encryption of m. We define the *advantage* of the adversary $\mathcal{A}$ against Π in these games as

$$\mathsf{Adv}^{\mathsf{ow-atk}}_{\Pi,\mathcal{A}}(t) = \left|\Pr\left[\ \mathsf{Exp}^{\mathsf{ow-atk}-0}_{\Pi,\mathcal{A}}(t) = 1\ \right] - \frac{1}{|\mathcal{M}|}\right|.$$

We say that Π is OW-CPA (resp. OW-CCA or OW-PCA) secure if $\mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi,\mathcal{A}}(t)$ (resp. $\mathsf{Adv}^{\mathsf{ow-cca}}_{\Pi,\mathcal{A}}(k)$ or $\mathsf{Adv}^{\mathsf{ow-pca}}_{\Pi,\mathcal{A}}(t)$) is negligible for every PPT $\mathcal{A}$.

De-randomizing IND-CPA Ciphers: Most post-quantum hybrid constructions go via a step of de-randomization of an IND-CPA cipher to obtain a rigid deterministic PRIV-CCA encryption scheme, see below. This is then combined with the Fujisaki-Okamoto transform to obtain the final hybrid cipher. However, the problem with this approach is that the de-randomization is a costly operation to perform in a threshold manner. Therefore our goal is to have a hybrid method which does not use this de-randomization. Nevertheless, one can use our hybrid construction with such de-randomizations if desired.

The de-randommization is via a process called by Bellare, Boldyreva and O'Neill [BBO07], Encrypt-with-Hash, which produces a rigid deterministic encryption scheme. Note, in [HHK17,JZC+18,JZM19] the authors reinvent the IND-CPA to OW-PCA transform called Encrypt-with-Hash from ten year earlier, in the context of deriving a post-quantum KEM-DEM construction via the Fujisaki-Okamoto transform in the QROM.

PRIV-CPA and PRIV-CCA Security For Deterministic Encryption: In [BBO07] security notions are presented for *deterministic* asymmetric encryption schemes. We simplify somewhat their definition for our purposes, in particular our PRIV adversaries will utilize only one challenge message and not a vector of messages. In what follows we let $\Pi = (\mathcal{DK}, \mathcal{DE}, \mathcal{DD})$ denote a deterministic public key encryption scheme with message space $\mathcal{DM}$.

An adversary $\mathcal{A} = (\mathcal{A}_m, \mathcal{A}_g)$ is a pair of PPT algorithms where $\mathcal{A}_m$ has single input 1^t and $\mathcal{A}_g$ has input $(1^t, c, \mathsf{pk})$; and no other inputs. The goal of $\mathcal{A}_m$ is to generate a message $m \in \mathcal{DM}$ and some side information s, whereas the goal of $\mathcal{A}_g$ is to take c (the encryption of m) and determine the side information. The algorithm $\mathcal{A}_m$ is the message generator and $\mathcal{A}_g$ is the information guesser. The adversary $\mathcal{A}$ is said to have *min-entropy* $\mu(t)$ if for all $x \in \mathcal{DM}$

$$\Pr\left[m = x : (m, s) \leftarrow \mathcal{A}_m(1^t)\right] \leq 2^{-\mu(t)}.$$

We say $\mathcal{A}$ has *high min-entropy* if $\mu(t) \in \omega(\log(t))$.

To formalize the attack notion we define the following experiments. Again we define two games; one where $\mathsf{atk} = \mathsf{cpa}$ in which the adversary $\mathcal{A}$ is given no additional helper oracles, and one where $\mathsf{atk} = \mathsf{cca}$ in which the adversary $\mathcal{A}_g$ (but not $\mathcal{A}_m$) is given a decryption oracle which on input of $c \neq c^*$ will return $\mathcal{DD}(\mathsf{sk}, c)$.

Experiment $\mathsf{Exp}^{\mathsf{priv-atk}-0}_{\Pi,\mathcal{A}}(t)$	Experiment $\mathsf{Exp}^{\mathsf{priv-atk}-1}_{\Pi,\mathcal{A}}(t)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{DK}(1^t)$	$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{DK}(1^t)$
$(m_0, s_0) \leftarrow \mathcal{A}_m(1^t)$	$(m_0, s_0) \leftarrow \mathcal{A}_m(1^t)$
$r \leftarrow \mathcal{R}$	$(m_1, s_1) \leftarrow \mathcal{A}_m(1^t)$
$c^* \leftarrow \mathcal{DE}(\mathsf{pk}, m_0; r)$	$r \leftarrow \mathcal{R}$
$g \leftarrow \mathcal{A}_g(1^t, c^*, \mathsf{pk})$	$c^* \leftarrow \mathcal{DE}(\mathsf{pk}, m_0; r)$
If $g = s_0$ then return one	$g \leftarrow \mathcal{A}_g(1^t, c^*, \mathsf{pk})$
Else return zero	If $g = s_1$ then return one
	Else return zero

We define the *advantage* of the adversary $\mathcal{A}$ against Π in this game as

$$\mathsf{Adv}^{\mathsf{priv-atk}}_{\Pi,\mathcal{A}}(t) = \left|\Pr\left[\ \mathsf{Exp}^{\mathsf{priv-atk}-0}_{\Pi,\mathcal{A}}(t) = 1\ \right] - \Pr\left[\ \mathsf{Exp}^{\mathsf{priv-atk}-1}_{\Pi,\mathcal{A}}(t) = 1\ \right]\right|.$$

We say that Π is PRIV-CPA (resp. PRIV-CCA) secure if $\mathsf{Adv}^{\mathsf{priv-cpa}}_{\Pi,\mathcal{A}}(t)$ (resp. $\mathsf{Adv}^{\mathsf{priv-cca}}_{\Pi,\mathcal{A}}(t)$) is negligible for every PPT $\mathcal{A}$. Note that PRIV-CPA (resp. PRIV-CCA) implies OW-CPA (resp. OW-CCA) since the adversary $\mathcal{A}_m$ can output (m, m).

Encrypt-with-Hash: The paper [BBO07] as well as providing the above security definition for deterministic public key encryption also gives the following construction, called *Encrypt-with-Hash*. The construction takes a *randomized* public key encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with space of random coins $\mathcal{R}$, a hash function G with codomain $\mathcal{R}$, and constructs a deterministic encryption scheme $\Pi' = (\mathcal{DK}, \mathcal{DE}, \mathcal{DD})$ as follows:

$\mathcal{DK}(1^t)$:	$\mathcal{DE}(\mathsf{pk}, m)$:	$\mathcal{DD}(\mathsf{sk}, c)$:
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^t)$	$r \leftarrow G(\mathsf{pk} \| m)$	$m \leftarrow \mathcal{D}(\mathsf{sk}, c)$
Return $(\mathsf{pk}, (\mathsf{sk}, \mathsf{pk}))$	$c \leftarrow \mathcal{E}(\mathsf{pk}, m; r)$	$r \leftarrow G(\mathsf{pk} \| m)$
	Return c	If $\mathcal{E}(\mathsf{pk}, m; r) = c$ then return m
		Else return $\perp$

If we define the *max public-key probability* $\mathsf{mpk}(t)$ of Π as

$$\mathsf{mpk}(t) = \max_w \Big(\Pr \Big[\ \mathsf{pk} = w \ : \ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^t) \ \Big] \Big).$$

We also define the *max ciphertext probability* $\mathsf{mc}(t)$ of Π as

$$\mathsf{mc}(t) = \max_{m \in \mathcal{M}} \Big(\Pr \Big[\ c_1 = c_2 \ : \ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^t), \ r_1, r_2 \leftarrow \mathcal{R}, \\ c_1 \leftarrow \mathcal{E}(\mathsf{pk}, m; r_1), \ c_2 \leftarrow \mathcal{E}(\mathsf{pk}, m; r_2) \ \Big] \Big).$$

Then we have the following result, from [BBO07],

Theorem 2.2. *Suppose there is a $\mathcal{A} = (\mathcal{A}_m, \mathcal{A}_g)$ PRIV-CCA adversary against Π' with minentropy μ making at most q_h queries to its hash oracle and q_d queries to its decryption oracle. Then there is an IND-CPA adversary B against Π such that*

$$\mathsf{Adv}^{\mathsf{priv-cca}}_{\Pi', \mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ind-cpa}}_{\Pi, \mathcal{B}} + \frac{2 \cdot q_h}{2^\mu} + 2 \cdot q_h \cdot \mathsf{mpk} + 2 \cdot q_d \cdot \mathsf{mc}.$$

Note, even the PRIV-CCA security of Π' rests solely on the IND-CPA security of Π. We obtain the result for PRIV-CPA by taking $q_d = 0$ in the above theorem.

Encryption With Distributed Decryption: Given a set $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ of parties, we consider access structures A consisting of a monotonically increasing set of subsets of $2^{\mathcal{P}}$. A set S is said to be qualified if $S \in A$, and unqualified otherwise. Given an encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ we say that the scheme admits a distributed decryption functionality for an access structure A, if there are two n-party protocols $\Pi_{\mathcal{K}}$ and $\Pi_{\mathcal{D}}$. The protocol $\Pi_{\mathcal{K}}$ produces some data sk_i for each party, called the secret key shares. The protocol $\Pi_{\mathcal{D}}$ on input of an agreed ciphertext c from all parties in $S \in A$, and the value sk_i from all parties in S, will output the value $m = \mathcal{D}(\mathsf{sk}, c)$.

The distributed decryption protocols are said to be secure (in the IND-ATK sense) if an unqualified set of adversarial parties cannot, while interacting with a qualified set of parties, break the IND-ATK security of the underlying encryption scheme. This security definition can be made more formal by saying that the distributed decryption protocol should act like an ideal decryption functionality. See [SG98,SG02] for a specific instantiation.

We shall assume an actively secure MPC protocol for the access structure A, and will then construct an algorithm which implements the algorithm $\mathcal{D}$ within the MPC protocol. Thus it automatically becomes a distributed protocol $\Pi_{\mathcal{D}}$ for the decryption functionality, and its security is inherited from the underlying MPC protocol. The challenging part is to develop the encryption scheme and the specific instantiation of $\mathcal{D}$ to enable the underlying MPC system to provide an efficient distributed implementation.

By using a generic MPC functionality, as opposed to a specific protocol, we restrict ourselves to the threshold case where *all* parties have to be involved in the computation; but where security is maintained against an adversary controlling a given threshold. This is in contrast to the models proposed in [SG98,SG02] which allow for a subset of the key-share holding parties to participate.

KEM-DEM Philosophy: A central tenant when using public key encryption in practice, is that one never encrypts a large message with a public key algorithm. Instead one encrypts the actual message with a fast symmetric key algorithm, such as AES-GCM, and then the symmetric key is

transferred to the recipient using a public key scheme. Thus producing a *hybrid* encryption scheme. In this way the symmetric key is only used once in the symmetric cipher, and thus we do not need a fully secure AEAD scheme but the weaker notion of a DEM, and the public key scheme is only needed to transport a single random key (and not a message) leading to the simpler public key construction of a KEM. See [CS03] for an extensive discussion, with the standard definitions and proofs.

We let $\Pi_p = (\mathcal{K}_p, \mathcal{E}_p, \mathcal{D}_p)$ denote an IND-CCA public key encryption scheme with message space $\mathcal{M}_p$, ciphertext space $\mathcal{C}_p$, and space of random coins $\mathcal{R}_p$, and let $\Pi_s = (\mathcal{K}_s, \mathcal{E}_s, \mathcal{D}_s)$ denote an IND-CCA symmetric key encryption scheme (which recall for us is always one-time and hence a DEM) with message space $\mathcal{M}_s = \{0,1\}^*$. From these two components one can construct a KEM-DEM encryption scheme for arbitrary long messages as follows: We first define a hash function $H : \mathcal{M}_p \longrightarrow \mathcal{K}_s$ where by abuse of notation by $\mathcal{K}_s$ we mean the key space of Π_s. We can then define a hybrid encryption scheme $\Pi_h = (\mathcal{K}_h, \mathcal{E}_h, \mathcal{D}_h)$ as follows

$\mathcal{K}_h(1^t)$:	$\mathcal{E}_h(\mathsf{pk}, m)$:	$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2))$:
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$	$k \leftarrow \mathcal{M}_p$	$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$
Return $(\mathsf{pk}, \mathsf{sk})$	$r \leftarrow \mathcal{R}_p, r' \leftarrow \mathcal{R}_s$	If $k = \perp$ then return $\perp$
	$\mathsf{k} \leftarrow H(k)$.	$\mathsf{k} \leftarrow H(k)$.
	$c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k; r)$	$m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
	$c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m; r')$	Return m
	Return (c_1, c_2)	

Naive Threshold KEM-DEM: The goal of our work is to produce threshold public key encryption for long messages; namely we would want to share the decryption key amongst a set of entities so that a given subset needs to come together to decrypt. Clearly we would not want the extra expense of the threshold decryption to impact when encrypting very large messages. Thus we would want to use something akin to the KEM-DEM philosophy, with the main message being encrypted and decrypted via a fast cipher such as AES-GCM.

Finding KEM-like constructions which admit distributed decryption protocols is relatively easy. However, whilst it is possible to execute AES in a threshold manner, see e.g. [KOR+17,PSSW09], the performance for long messages is prohibitive. Thus distributed DEMs are much harder to obtain. For this reason we would like to apply the decryption of the large message 'in the clear', but this implies that the decryption algorithm will need to 'leak' the decryption key k of the DEM component. In particular this key will leak irrespective of whether the DEM decrypts correctly or not; since the decrypting parties need to obtain the DEM key before it is known whether the key is valid for the DEM.

Our decryption algorithm functionality, and thus the functionality of any decryption oracle given to an adversary, would therefore be of the form

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2))$:
$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$
If $k = \perp$ then return $(\perp, \perp)$
$\mathsf{k} \leftarrow H(k)$.
$m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
Return (k, m)

This provides an immediate attack in the standard IND-CCA model on the hybrid construction. An adversary takes the target ciphertext (c_1^*, c_2^*), submits (c_1^*, c_2) to the decryption oracle for a

random value c_2. With high probability, they will receive $(k, \perp)$. Then, they can use k to obtain k and thus decrypt c_2^*, and so win the security game. It is to avoid this attack that we modify the KEM-DEM framework in the next section.

Generic Multi-Party Computation Our methodology uses a *generic* actively-secure-with-abort MPC functionality defined via Linear Secret Sharing over the field $\mathbb{F}_q$. This means that inputs of the parties remain private throughout the execution of the protocol, and when a set of adversaries deviate from the protocol, honest parties will catch this with overwhelming probability and then abort from the protocol. This should be compared to passively secure protocols which offer a much weaker guarantee that security is only preserved if all parties follow the precise protocol steps correctly. We present in Figure 1 the base MPC functionality. Despite using a generic underlying protocol, our protocol ends up being surprisingly efficient. This is because we carefully designed Gladius to be both efficient in a distributed and a non-distributed manner.

Operations for Secure Computation, $\mathcal{F}_{\mathsf{MPC}}$.

The functionality runs with $\mathcal{P} = \{\mathsf{P}^1, \ldots, \mathsf{P}^n\}$ and an ideal adversary $\mathcal{A}$, that statically corrupts a set A of parties. Given a set I of valid identifiers, all values are stored in the form $(varid, x)$, where $varid \in I$.

Initialize: On input $(init, p)$ from all parties, the functionality stores $(domain, p)$,

Input: On input $(input, \mathsf{P}^i, varid, x)$ from P^i and $(input, \mathsf{P}^i, varid, ?)$ from all other parties, with $varid$ a fresh identifier, the functionality stores $(varid, x)$.

Add: On command $(add, varid_1, varid_2, varid_3)$ from all parties (if $varid_1, varid_2$ are present in memory and $varid_3$ is not), the functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x + y)$.

Multiply: On input $(multiply, varid_1, varid_2, varid_3)$ from all parties (if $varid_1, varid_2$ are present in memory and $varid_3$ is not), the functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x \cdot y)$.

Output: On input $(output, varid, i)$ from all honest parties (if $varid$ is present in memory), the functionality retrieves $(varid, y)$ and outputs it to the environment. The functionality waits for an input from the environment. If this input is $\mathsf{Deliver}$ then y is output to all players if $i = 0$, or y is output to player i if $i \neq 0$. If the adversarial input is not equal to $\mathsf{Deliver}$ then all player abort.

Figure 1. Operations for Secure Computation, $\mathcal{F}_{\mathsf{MPC}}$.

To ease notation we denote a variable $x \in \mathbb{F}_q$ stored within the MPC functionality via $\langle x \rangle$, and write addition and multiplication of shares as $\langle x \rangle + \langle y \rangle$ and $\langle x \rangle \cdot \langle y \rangle$. We extend the notation to vectors and matrices in the obvious way via $\langle \mathbf{x} \rangle$ and $\langle A \rangle$. If $\langle \mathbf{x} \rangle$ is a shared vector we let $\langle x_i \rangle$ denote the shared entries, and if $\langle A \rangle$ is a shared matrix we let $\langle A^{(i,j)} \rangle$ denote the shared entries; with a similar notation for vectors and matrices of non-shared values.

The cost model for LSSS-based MPC protocols is such that addition of such shared entities is 'for free', whereas multiplication consumes resources (typically communication). Many MPC protocols in this setting, such as [BDOZ11,DPSZ12,KOS16,SW19], work in an offline/online manner. In this setting the multiplication not only consumes communication resources in the online phase, but also consumes some correlated randomness (so-called Beaver triples) from the offline phase. However, an advantage of these offline/online models is that one can prepare other forms of correlated randomness in the offline phase; such as shares of random bits $\langle b \rangle$ with an unknown $b \in \{0, 1\}$. In our algorithms below we will write this as $\langle b \rangle \leftarrow \mathsf{Bits}()$. If we sample a shared random element in $\mathbb{F}_q$, we will denote this by $\langle x \rangle \leftarrow \mathbb{F}_q$. To open an element we will write $x \leftarrow \mathbf{Output}(\langle x \rangle)$ when it is output to all players.

MPC Friendly Hash Functions: Rescue: Our LWR-based construction of a hybrid cipher with efficient distributed decryption will make use of an MPC-friendly hash function, such as those in [AAB+19,GKK+19]. These hash function constructions are sponge-based, and there are two types; those suitable for MPC over characteristic two fields (StarkAD and Vision) and those suitable for MPC over large prime fields (Poseidon and Rescue). In this paper, we concentrate on the Rescue design from [AAB+19], which seems more suited to our application.

Rescue has a state of $t = r + c$ finite field elements in $\mathbb{F}_q$. The initial state of the sponge is defined to be the vector of t zero elements. A message is first mapped into $n = d \cdot r$ elements in $\mathbb{F}_q$, $m_0, m_1, \ldots, m_{n-1}$. The elements are absorbed into the sponge in d absorption phases, where r elements are absorbed in each phase. At each phase a permutation $f : \mathbb{F}_q^t \longrightarrow \mathbb{F}_q^t$ is applied resulting in a state $s_0, \ldots, s_{t-1}$. At the end the absorption the r values $s_c, \ldots, s_{t-1}$ are output from the state. This process can then be repeated, with more data absorbed and then squeezed out. Thus we are defining a map $H : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^r$.

Each primitive call f in the Rescue sponge is performed by executing a round function rnds times. The round function is parametrized by a (small prime) value α, an MDS matrix $M \in \mathbb{F}_q^{t \times t}$ and two step constants $\mathbf{k}_i, \mathbf{k}'_i \in \mathbb{F}_q^t$. The value α is chosen to be the smallest prime such that $\gcd(q-1, \alpha) = 1$. The round function is consists in apply exponentiation by $1/\alpha$, followed by application of the MDS matrix, followed by addition of the round constant $\mathbf{k}_i$, followed by exponentiation by α, followed by a further application of the MDS matrix, followed by addition of the round constant $\mathbf{k}'_i$. To obtain security of the entire construction we require that

$$\begin{aligned}
\ell_0 &= \left\lceil \frac{2 \cdot \kappa}{\log_2(q^{t+1}) - \log_2((\alpha-1)^{t+1})} \right\rceil, \\
\ell_1 &= \begin{cases} \left\lceil \frac{\kappa+2}{4 \cdot t} \right\rceil & \alpha = 3, \\ \left\lceil \frac{\kappa+3}{5.5 \cdot t} \right\rceil & \alpha \geq 5, \end{cases} \\
\mathsf{rnds} &= 2 \cdot \max(\ \ell_0\ ,\ \ell_1\ ,\ 5\ ), \\
\min(r, c) &\geq 2 \cdot \kappa / \log_2 q
\end{aligned}$$

where κ is the desired security parameter. See [BST19] for a discussion of implementing Rescue in an MPC system; albeit for a large prime characteristic q of more than 256-bits. In our application q will be in the region of 21-bits.

3 Generic Hybrid Constructions

We let $\Pi_p = (\mathcal{K}_p, \mathcal{E}_p, \mathcal{D}_p)$ denote a OW-CPA secure, rigid, deterministic (resp. a OW-PCA secure, rigid and randomized) public key encryption scheme with message space $\mathcal{M}_p$ and ciphertext space $\mathcal{C}_p$ which is OW-CPA secure, We let $\Pi_s = (\mathcal{K}_s, \mathcal{E}_s, \mathcal{D}_s)$ denote a (one-time) IND-CPA symmetric key encryption scheme with message space $\mathcal{M}_s = \{0,1\}^*$ and ciphertext space $\mathcal{C}_s \subset \{0,1\}^*$. Again by abuse of notation we let $\mathcal{K}_s$ also denote the key space of Π_s.

3.1 Hybrid$_1$ Construction

For this construction we define two hash functions

$$H : \mathcal{M}_p \longrightarrow \mathcal{K}_s,$$

$$G : \begin{cases} \{0,1\}^* \times \mathcal{M}_p \longrightarrow \{0,1\}^{|G|} & \text{If } \Pi_p \text{ is deterministic} \\ \mathcal{C}_p \times \{0,1\}^* \times \mathcal{M}_p \longrightarrow \{0,1\}^{|G|} & \text{If } \Pi_p \text{ is randomized} \end{cases}$$

Note, G is defined to take elements in $\mathcal{M}_p$ as the last entry for efficiency reasons (see below). We can then define our first hybrid encryption scheme $\Pi_h = (\mathcal{K}_h, \mathcal{E}_h, \mathcal{D}_h)$ as follows

$\mathcal{K}_h(1^t)$:	$\mathcal{E}_h(\mathsf{pk}, m)$:	$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3))$:
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$		
Return $(\mathsf{pk}, \mathsf{sk})$	$k \leftarrow \mathcal{M}_p$	$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$
	$\mathsf{k} \leftarrow H(k)$	If $k = \perp$ then return $(\perp, \perp)$.
	$r \leftarrow \mathcal{R}_s$	$t \leftarrow G(c_2, k)$
	$c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k)$	(resp. $t \leftarrow G(c_1, c_2, k)$)
	$c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m; r)$	If $t \neq c_3$ then return $(\perp, \perp)$.
	$c_3 \leftarrow G(c_2, k)$	$\mathsf{k} \leftarrow H(k)$.
	(resp. $c_3 \leftarrow G(c_1, c_2, k)$)	$m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
	Return (c_1, c_2, c_3)	Return (k, m).

Notice how the decryption function 'leaks' the key k which is encrypted by the deterministic function even when the decryption function $\mathcal{D}_s$ fails. This will allow us, in our threshold decryption operation, to also leak this key before the algorithm $\mathcal{D}_s$ is called, enabling $\mathcal{D}_s$ to be applied in the clear. The only question though is whether leaking this key is secure. The attack described from the last section does not apply, as the invalid ciphertext is already rejected by the testing for the correct value of G, which does not leak k if the test fails. In what follows we call this check the G-check.

As remarked in the introduction the variant of the hybrid construction which utilizes a determinsitic Π_p can be seen as a special form of the second Fujisaki-Okamoto hybrid construction; assuming the space $\mathcal{M}_p$ is exponentially large to ensure the resulting 'randomized' public key scheme is well-spread. Thus, the above hybrid construction is secure not only in the ROM, but also in the QROM, when we do not leak the secret key k during the decryption process and when Π_p is perfectly correct.

In the standard random oracle model, the following theorem shows that first hybrid construction is secure in a model in which the key k does leak during decryption as above, and where we combine it with a generic one-time IND-CPA DEM.

Theorem 3.1. *If H and G are modelled as random oracles then if $\mathcal{A}$ is an IND-CCA adversary against Π_h then there is an OW-CPA adversary (resp. OW-PCA) adversary $\mathcal{B}$ against the deterministic (resp. randomized) rigid public key scheme Π_p, which is δ_c-Collision Free, a (one-time) IND-CPA adversary $\mathcal{C}$ against Π_s, and a $\delta - \mathsf{Aware}$ adversary $\mathcal{D}$ against Π_p such that*

$$\mathsf{Adv}^{\mathsf{ind-cca}}_{\Pi_h,\mathcal{A}}(t) \leq \mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi_p,\mathcal{B}}(t) + \mathsf{Adv}^{\mathsf{ind-cpa}}_{\Pi_s,\mathcal{C}}(t) + q_d \cdot \mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi_p,\mathcal{D}}(t) + \frac{1}{|\mathcal{M}_p|} + \frac{2 \cdot q_d + q_G^2}{2^{|G|}} + \delta_c$$

where q_d (resp. q_G) is an upper bound on the number of decryption oracle (resp. G-oracle) queries and the decryption oracle queries made to the hybrid scheme leak the key k as above.

Proof. Let $\mathcal{A}$ be an adversary against the scheme Π_h above in the IND-CCA sense. Recall $\mathcal{A}$ runs in two phases: In phase one $\mathcal{A}_1$ has input the public key pk and it returns two messages m_0 and

m_1, of equal length, plus some state information state. The target ciphertext, which in $\mathsf{Exp}^{\mathsf{ind-cca}-b}_{\Pi_h,\mathcal{A}}$ encrypts the message m_b, we will denote by $c^* = (c_1^*, c_2^*, c_3^*)$. This is constructed as follows:

$$
\begin{aligned}
&k^* \leftarrow \mathcal{M}_p \\
&\mathsf{k}^* \leftarrow H(k^*) \\
&r^* \leftarrow \mathcal{R}_s \\
&c_1^* \leftarrow \begin{cases} \mathcal{E}_p(\mathsf{pk}, k^*) & \text{If } \Pi_p \text{ is deterministic} \\ \mathcal{E}_p(\mathsf{pk}, k^*; r') & \text{For } r' \leftarrow \mathcal{R}_p \text{ if } \Pi_p \text{ is randomized} \end{cases} \\
&c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r^*) \\
&c_3^* \leftarrow \begin{cases} G(c_2^*, k^*) & \text{If } \Pi_p \text{ is deterministic} \\ G(c_1^*, c_2^*, k^*) & \text{If } \Pi_p \text{ is randomized} \end{cases}
\end{aligned}
$$

We define three bad events:

- Let bad_1 denote the event that the adversary makes a query to its decryption oracle for a ciphertext (c_1, c_2, c_3) such that c_1 is a ciphertext for which there are two messages which map to it.
- Let bad_2 denote the event that the adversary queries the decryption oracle for a ciphertext (c_1, c_2, c_3) such that $c_1 = \mathcal{E}_p(\mathsf{pk}, k)$ with $\mathcal{D}_p(\mathsf{sk}, c) = \perp$ and in addition queries G on an input (c_2', k) (resp. (c_1', c_2', k)) for the same value of k (but possibly different c_1' and c_2').
- Let bad_3 denote the event that $\mathcal{A}$ calls the random oracle H on input k^* or $\mathcal{A}$ calls the random oracle G on input (c_2, k^*) (resp. (c_1, c_2, k^*)) for an arbitrary value c_2 (resp. two arbitrary values c_1 and c_2).

We thus have, setting $\mathsf{bad} = \mathsf{bad}_1 \ \vee \ \mathsf{bad}_2 \ \vee \mathsf{bad}_3$,

$$
\begin{aligned}
&\left| \Pr\left[\ \mathsf{Exp}^{\mathsf{ind-cca}-0}_{\Pi_h,\mathcal{A}}(t) = 1 \ \right] - \Pr\left[\ \mathsf{Exp}^{\mathsf{ind-cca}-1}_{\Pi_h,\mathcal{A}}(t) = 1 \ \right] \right| \\
&\qquad \leq \Pr[\ \mathsf{bad}\ ] + \left| \Pr\left[\ \mathsf{Exp}^{\mathsf{ind-cca}-0}_{\Pi_h,\mathcal{A}}(t) = 1 \mid \neg\mathsf{bad} \ \right] \right. \\
&\qquad\qquad \left. - \Pr\left[\ \mathsf{Exp}^{\mathsf{ind-cca}-1}_{\Pi_h,\mathcal{A}}(t) = 1 \mid \neg\mathsf{bad} \ \right] \right|
\end{aligned}
$$

To bound the second term in this inequality we define two new experiments $\mathsf{Exp}^{\mathsf{ind'-cca}-b}_{\Pi_h,\mathcal{A}}$ for $b \in \{0,1\}$ where we replace the above construction of the challenge ciphertext by the following construction, where the key k^* is now sampled at random and has nothing to do with k^*

$$
\begin{aligned}
&k^* \leftarrow \mathcal{M}_p \\
&\mathsf{k}^* \leftarrow \mathcal{K}_s \\
&r^* \leftarrow \mathcal{R}_s \\
&c_1^* \leftarrow \begin{cases} \mathcal{E}_p(\mathsf{pk}, k^*) & \text{If } \Pi_p \text{ is deterministic} \\ \mathcal{E}_p(\mathsf{pk}, k^*; r') & \text{For } r' \leftarrow \mathcal{R}_p \text{ if } \Pi_p \text{ is randomized} \end{cases} \\
&c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r^*) \\
&c_3^* \leftarrow \begin{cases} G(c_2^*, k^*) & \text{If } \Pi_p \text{ is deterministic} \\ G(c_1^*, c_2^*, k^*) & \text{If } \Pi_p \text{ is randomized} \end{cases}
\end{aligned}
$$

It is clear that we have

$$\Big|\Pr\Big[\ \mathsf{Exp}^{\mathsf{ind-cca}-0}_{\Pi_h,\mathcal{A}}(t)=1 \mid \neg\mathsf{bad}\ \Big] - \Pr\Big[\ \mathsf{Exp}^{\mathsf{ind-cca}-1}_{\Pi_h,\mathcal{A}}(t)=1 \mid \neg\mathsf{bad}\ \Big]\Big| \\ = \Big|\Pr\Big[\ \mathsf{Exp}^{\mathsf{ind'-cca}-0}_{\Pi_h,\mathcal{A}}(t)=1 \mid \neg\mathsf{bad}\ \Big] - \Pr\Big[\ \mathsf{Exp}^{\mathsf{ind'-cca}-1}_{\Pi_h,\mathcal{A}}(t)=1 \mid \neg\mathsf{bad}\ \Big]\Big|.$$

We aim to show that given an adversary $\mathcal{A}$ which tries to distinguish between $\mathsf{Exp}^{\mathsf{ind'-cca}-0}_{\Pi_h,\mathcal{A}}$ and $\mathsf{Exp}^{\mathsf{ind'-cca}-1}_{\Pi_h,\mathcal{A}}$ (assuming $\neg\mathsf{bad}$) we can construct an adversary $\mathcal{C}$ which tries to distinguish between $\mathsf{Exp}^{\mathsf{ind-cpa}-0}_{\Pi_s,\mathcal{C}}$ and $\mathsf{Exp}^{\mathsf{ind-cpa}-1}_{\Pi_s,\mathcal{C}}$. In other words

$$\Big|\Pr\Big[\ \mathsf{Exp}^{\mathsf{ind'-cca}-0}_{\Pi_h,\mathcal{A}}(t)=1 \mid \neg\mathsf{bad}\ \Big] - \Pr\Big[\ \mathsf{Exp}^{\mathsf{ind'-cca}-1}_{\Pi_h,\mathcal{A}}(t)=1 \mid \neg\mathsf{bad}\ \Big]\Big| \\ \leq \mathsf{Adv}^{\mathsf{ind-cpa}}_{\Pi_s,\mathcal{C}}(t) + \frac{q_d}{2^{|G|}}. \tag{1}$$

Algorithm $\mathcal{C}_1$ is constructed as follows. It generates a public/private key pair $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{K}_p$ and passed pk to algorithm $\mathcal{A}_1$. When $\mathcal{A}_1$ returns with the tuple $(m_0, m_1, \mathsf{state})$, Algorithm $\mathcal{C}_1$ returns $(m_0, m_1, (\mathsf{pk},\mathsf{sk},\mathsf{state}))$ to its environment. Algorithm $\mathcal{C}_2$ is then called with input $(c^*, (\mathsf{pk},\mathsf{sk},\mathsf{state}))$. At this point $\mathcal{C}_2$ needs to create a ciphertext to pass to algorithm $\mathcal{A}_2$; which it does as follows: It generates a $k^* \leftarrow \mathcal{M}_p$ and computes $c_1^* = \mathcal{E}_p(\mathsf{pk}, k^*)$, and sets $c_2^* \leftarrow c^*$ and c_3^* at random from the codomain of G. The tuple $((c_1^*, c_2^*, c_3^*), \mathsf{state})$ is returned to algorithm $\mathcal{A}_2$. Eventually algorithm $\mathcal{A}_2$ returns a bit b which algorithm $\mathcal{C}_2$ returns to its environment.

The oracle calls of $\mathcal{A}_1$ and $\mathcal{A}_2$ are handled as follows:

- $G(c_2, k)$ (resp. $G(c_1, c_2, k)$): This is processed in the standard way, and since event bad does not happen it never has to answer any query on k^*.
- $H(k)$: This is processed in the standard way. Again as event bad does not happen it never has to respond with $H(k^*)$.
- $\mathcal{O}_{\mathcal{D}_h}(c_1, c_2, c_3)$: Decryption oracle queries are handled as follows:
  - If $c_1 \neq c_1^*$ then decrypt c_1 using $\mathcal{D}_p(\mathsf{sk}, c_1)$ and proceed as for normal decryption.
  - If $c_1 = c_1^*$ and $c_2 = c_2^*$ then output $(\perp, \perp)$. Since either $c_3 = c_3^*$ in which case this call is invalid, or $c_3 \neq c_3^*$ in which case the valid decryption oracle will reject this ciphertext via the G-check.
  - If $c_1 = c_1^*$ and $c_2 \neq c_2^*$ then since bad does not happen we know the adversary has not called G on input (c_2, k^*) (resp. (c_1^*, c_2, k^*)) (and never will do) thus we can output $(\perp, \perp)$. This will be the correct response unless the oracle G would respond with c_3, which happens with probability $1/2^{|G|}$. This corresponds to the $q_d/2^{|G|}$ term in equation (1).

It is easy to see that this simulation is perfect on the assumption that event bad does not happen.

We now turn to bounding $\Pr[\ \mathsf{bad}\ ] = \Pr[\mathsf{bad}_1 \ \vee\ \mathsf{bad}_2\ \vee \mathsf{bad}_3]$. By assumption $\Pr[\mathsf{bad}_1] \leq \delta_c$, thus we have

$$\Pr[\ \mathsf{bad}\ ] \leq \delta_c + \Pr[\neg\mathsf{bad}_1\ \wedge\ (\ \mathsf{bad}_2\ \vee \mathsf{bad}_3\ )].$$

To bound the last term we simulate the hash and decryption oracles as follows (the hash oracle queries are handled as standard random oracle queries):

- We search for (k, h) on the H-List, if such an entry exists it returns h, otherwise it generates h at random, adds (k, h) to the H-List and returns h.

- When $G(c_2, k)$ (resp. $G(c_1, c_2, k)$) is called we searches for $((c_2, k), g)$ (resp. $((c_1, c_2, k), g)$) on the G-List, if such an entry exists it returns g, otherwise it generates g at random, adds $((c_2, k), g)$ (resp. $((c_1, c_2, k), g)$) to the G-List and returns g. If ever a collision in output values of the G oracle occurs then abort; which happens with probability roughly $q_g^2/2^{|G|}$.
- When $\mathcal{A}_1$ makes a decryption oracle query (c_1, c_2, c_3) in the case of Π_p being deterministic, the simulator check whether there is an entry $((c_2', k), g')$ on the G-List such that c_1 is an encryption of k (using the fact that Π_p is deterministic encryption). Since Π_p is rigid if c_1 decrypts to anything then then that something must encrypt to c_1. If there is no such entry then $\mathcal{B}$ returns $(\perp, \perp)$, which will be an invalid response with probability $1/2^{|G|}$ as the probability of G returning c_3 is $1/2^{|G|}$. This accounts for a $q_d/2^{|G|}$ term in the theorem.
  Since event bad_1 does not happen the ciphertext c_1 cannot be the encryption of any other value other than k. We proceed assuming that c_1 encrypts k, but does not decrypt to $\perp$.
  Note that $((c_2', k), g')$ is on the G-List, but not necessarily $((c_2, k), g)$ at this point. Thus if there is no entry of the form $((c_2, k), g)$ on the G-List we can simulate the query using the simulation above. We can thus assume that $((c_2, k), g)$ lies in the G-List for some g. If $g \neq c_3$ then $\mathcal{B}$ returns $(\perp, \perp)$, since such a decryption query will result in the G-check failing.
  At this point we know the G-Check will pass and so we can simulate a call $\mathsf{k} = H(k)$ as above, and proceeds to attempt to decrypt c_2 using k, obtaining either $r = \perp$ or $r = m$, a valid message. In both cases algorithm $\mathcal{B}$ replies with (k, r).
- When $\mathcal{A}_1$ makes a decryption oracle query (c_1, c_2, c_3) in the case of Π_p being randomized, we check whether there is an entry $((c_1', c_2', k), g')$ on the G-List such that c_1 is an encryption of k, (this is done using the PCA oracle provided in this case). Again since Π_p is rigid we know that if c_1 decrypts to k, then there is a randomness choice in encryption so that k encrypts to c_1. If there is no such entry then $\mathcal{B}$ returns $(\perp, \perp)$, which will be an invalid response with probability $1/2^{|G|}$ as the probability of G returning c_3 is $1/2^{|G|}$. This accounts for a $q_d/2^{|G|}$ term in the theorem.
  As before we now know there is an entry of the form $((c_1', c_2', k), g')$ on the G-List such that k encrypts to c_1, and that c_1 decrypts to k.
  For this value of k if there is no entry of the form $((c_1, c_2, k), g)$ on the G-List then $\mathcal{B}$ simulates the query using the simulation above, we can thus assume that $((c_1, c_2, k), g)$ lies in the G-List for some g. If $g \neq c_3$ then $\mathcal{B}$ returns $(\perp, \perp)$, since such a decryption query will result in the G-check failing.
  At this point we know the G-Check will pass and so algorithm $\mathcal{B}$ simulates a call $\mathsf{k} = H(k)$ as above, and proceeds to attempt to decrypt c_2 using k, obtaining either $r = \perp$ or $r = m$, a valid message. In both cases algorithm $\mathcal{B}$ replies with (k, r).

This simulation will be perfect unless event bad_2 happens. In that case we can use the simulation to construct an algorithm $\mathcal{D}$ which runs the above simulation and simply return a pair (k, c_1) for a c_1 which was passed to the decryption oracle which it decrypted to k. With probability $1/q_d$ this will correspond to the bad_2 event and will be a solution to the $\perp$-Aware problem. Thus we have

$$\begin{aligned}
\Pr[\ \mathsf{bad}\ ] &\leq \delta_c + \Pr[\neg\mathsf{bad}_1\ \wedge\ (\ \mathsf{bad}_2\ \vee\ \mathsf{bad}_3\ )] \\
&\leq \delta_c + \Pr[\neg\mathsf{bad}_1\ \wedge\ \mathsf{bad}_2] + \Pr[\neg\mathsf{bad}_1\ \wedge\ \neg\mathsf{bad}_2\ \wedge\ \mathsf{bad}_3] \\
&\leq \delta_c + \frac{q_d + q_g^2}{2^{|G|}} + q_d \cdot \mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi_p, \mathcal{D}}(t) + \Pr[\neg\mathsf{bad}_1\ \wedge\ \neg\mathsf{bad}_2\ \wedge\ \mathsf{bad}_3].
\end{aligned}$$

The main part of the proof is to show that

$$\Pr[\neg \mathsf{bad}_1 \ \wedge \ \neg \mathsf{bad}_2 \ \wedge \ \mathsf{bad}_3]] \leq \mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi_p,\mathcal{B}}(t) + \frac{1}{|\mathcal{M}_p|}$$

for some adversary $\mathcal{B}$. Adversary $\mathcal{B}$ has as input a public key pk and a target ciphertext c^* for the public key scheme $(\mathcal{K}_p, \mathcal{E}_p, \mathcal{D}_p)$. Algorithm $\mathcal{B}$ calls algorithm $\mathcal{A}_1$, responding to the oracle queries as above, but with the following important modifications:

- When $H(k)$ is called algorithm $\mathcal{B}$ checks whether c^* is an encryption of k under pk. It can do this since $\mathcal{E}_p$ is deterministic (alternatively by using the PCA oracle in the case when $\mathcal{E}_p$ is not deterministic). If c^* is an encryption of k then $\mathcal{B}$ terminates and outputs k.
- When $G(c_2, k)$ (resp. $G(c_1, c_2, k)$) is called algorithm $\mathcal{B}$ checks whether c^* is an encryption of k under pk, as above. If c^* is an encryption of k then $\mathcal{B}$ terminates and outputs k.

At some point $\mathcal{A}_1$ returns two messages m_0 and m_1. Algorithm $\mathcal{B}$ forms an encryption of m_b by setting $c_1^* = c^*$, generating k^* at random, computing $c_2^* = \mathcal{E}_s(\mathsf{k}^*, m_b)$, and finally generating c_3^* at random. Thus we have implicitly defined $H(k^*) = \mathsf{k}^*$ and $G(c_1^*, c_2^*, k^*) = c_3^*$, for the unknown value k^* which is encrypted by c_1^*. The ciphertext (c_1^*, c_2^*, c_3^*) is now returned to algorithm $\mathcal{A}_2$. Algorithm $\mathcal{A}_2$ proceeds, with the oracle queries being answered by $\mathcal{B}$ as above. We need to modify the decryption oracle in the following order of conditionals:

- The query (c_1^*, c_2^*, c_3^*) is not allowed.
- The query (c_1^*, c_2^*, c_3) will result in $(\perp, \perp)$ since the G-Check will fail in both instances.
- The query (c_1^*, c_2, c_3^*) will result in $(\perp, \perp)$ being output, as the only way this could be a valid query is if a collision in G was found.
- The query (c_1^*, c_2, c_3) is processed as before which will either lead to algorithm $\mathcal{A}$ terminating by outputting the key k^* or the output of $(\perp, \perp)$.
- The query (c_1, c_2^*, c_3^*) in the case of a deterministic Π_p can only be valid if a collision in G is found. Since c_1 will decrypt to $k \neq k^*$ and we would have $G(c_2^*, k) = G(c_2^*, k^*) = c_3^*$. The same is true for the case of the randomized scheme, but only because we included c_1 into the domain of G in this case, since in this case c_1 could decrypt to k^*.
- The query (c_1, c_2^*, c_3) for $c_3 \neq c_3^*$ is processed as before.
- The query (c_1, c_2, c_3^*) will result in $(\perp, \perp)$ being output, as the only way this could be a valid query is if a collision in G was found.
- In all other cases it is processed as before.

If we assume event bad_3 happens then algorithm $\mathcal{B}$ will terminate and output the valid message k^* for which c^* is an encrypton. In which case this is a valid solution to the OW-CPA problem. □

3.2 Hybrid$_2$ Construction

Our second hybrid construction focuses solely on the case of Π_p being a rigid deterministic OW-CPA public key encryption scheme, we show that the generic hybrid transform, which uses the four hash functions,

$$\begin{aligned} H &: \mathcal{M}_p \longrightarrow \mathcal{K}_s, \\ H', H'' &: \mathcal{M}_p \longrightarrow \mathcal{M}_p \\ G &: \{0,1\}^* \times \mathcal{M}_p \longrightarrow \{0,1\}^{|G|} \end{aligned}$$

given by

$\mathcal{K}_h(1^t)$:	$\mathcal{E}_h(\mathsf{pk}, m)$:	$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4))$:
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$	$k \leftarrow \mathcal{M}_p$	
Return $(\mathsf{pk}, \mathsf{sk})$	$\mathsf{k} \leftarrow H(k)$	$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$
	$\mu \leftarrow H'(k)$	If $k = \perp$ then return $(\perp, \perp)$.
	$r \leftarrow \mathcal{R}_s$	$t \leftarrow H''(k)$
	$c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k)$	If $t \neq c_4$ then return $(\perp, \perp)$.
	$c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m; r)$	$\mu \leftarrow H'(k)$
	$c_3 \leftarrow G(c_2, \mu)$	$t' \leftarrow G(c_2, \mu)$
	$c_4 \leftarrow H''(k)$	If $t' \neq c_3$ then return $(\perp, \perp)$.
	Return (c_1, c_2, c_3, c_4)	$\mathsf{k} \leftarrow H(k)$.
		$m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
		Return (k, m).

is secure in the QROM. Namely we have the following theorem

Theorem 3.2. *If G, H, H' and H'' are modelled as quantum random oracles then if $\mathcal{A}$ is an IND-CCA adversary against Π_h then there is a (one-time) IND-CPA adversary $\mathcal{B}$ against Π_s, a $\perp$-Aware adversary $\mathcal{C}$ against the deterministic rigid public key scheme Π_p – which is δ_c-Collision Free and δ being the probability of its decryption failure for a uniformly random message – and OW-CPA adversaries $\mathcal{D}$ and $\mathcal{E}$ against Π_p such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ind-cca}}_{\Pi_h,\mathcal{A}}(t) \leq\ & \mathsf{Adv}^{\mathsf{ind-cpa}}_{\Pi_s,\mathcal{B}}(t) \ +\ \delta \\ & + 4q_1\sqrt{\frac{q_3}{\sqrt{|\mathcal{M}_p|}} + \frac{q_d}{2^{|G|}} + \delta' + \mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi_p,\mathcal{D}}(t) + 2q_2\sqrt{\delta' + \mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi_p,\mathcal{E}}(t)}}\end{aligned}$$

for $q_1 = q_H + q_{H'} + 2q_d$, $q_2 = q_{H''} + q_d$, $q_3 = 2(q_G + q_d + 1)$ and

$$\delta' = \delta_c + q_d \cdot \mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi_p,\mathcal{C}}(t) + \frac{1}{|\mathcal{M}_p|},$$

where q_d, q_G, q_H, $q_{H'}$ and $q_{H''}$ are respective upper bounds on the number of decryption oracle, G-oracle, H-oracle, H'-oracle and H''-oracle queries and the decryption oracle queries made to the hybrid scheme leak the key k as above.

First, we introduce some lemmas in the QROM that will be used to prove this theorem. The following lemma allows a *perfect* simulation of a quantum random oracle against an adversary.

Lemma 3.1 (Simulating a QRO [Zha12]). *Let $H(.)$ be an oracle drawn from the set of $2q$-wise independent functions uniformly at random. Then the advantage any quantum algorithm making at most q quantum queries to $H(.)$ has in distinguishing $H(.)$ from a truly random oracle is identically zero.*

The second lemma intuitively states that a quantum random oracle can be used as a *quantum-accessible* pseudo-random function, even if the distinguisher is given full access to the quantum random oracle in addition to the PRF oracle.

Lemma 3.2 (PRF based on a QRO). *Let Ω_H be the set of all functions $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ and Ω_R be the set of all functions $R : \mathcal{X} \to \mathcal{Y}$. Let $H \xleftarrow{\$} \Omega_H$, $k \xleftarrow{\$} \mathcal{K}$ and $R \xleftarrow{\$} \Omega_R$. Define the oracles $F_0 = H(k, .)$ and $F_1 = R(.)$. Consider an oracle algorithm/distinguisher A^{H,F_i} that makes at most*

q queries to H and F_i ($i \in \{0,1\}$). If ("the PRF key") k is chosen independently from A^{H,F_i}'s view, then we have

$$|\Pr[1 \leftarrow A^{H,F_0}] - \Pr[1 \leftarrow A^{H,F_1}]| \;\leq\; \frac{2q}{\sqrt{|\mathcal{K}|}}$$

The third lemma provides a generic reduction from a hiding-style property (indistinguishability) to a one-wayness-style property (unpredictability) in the QROM.

Lemma 3.3 (One-Way to Hiding (OW2H) [Unr14]). *Let Ω_H be the set of all functions $H : \mathcal{X} \to \mathcal{Y}$ and let $H \xleftarrow{\$} \Omega_H$ be a quantum random oracle. Consider an oracle algorithm A^H that makes at most q queries to H. Let B^H be an oracle algorithm that on input x does the following: picks $i \xleftarrow{\$} \{1,\ldots,q\}$ and $y \xleftarrow{\$} \mathcal{Y}$, runs $A^H(x,y)$ until (just before) the i-th query, measures the argument of the query in the computational basis and outputs the measurement outcome (if A makes less than i queries, B outputs $\perp \notin \mathcal{X}$). Let,*

$$P_A^1 = \Pr[b' = 1 : H \xleftarrow{\$} \Omega_H,\, x \xleftarrow{\$} \mathcal{X},\, b' \leftarrow A^H(x, H(x))]$$
$$P_A^2 = \Pr[b' = 1 : H \xleftarrow{\$} \Omega_H,\, x \xleftarrow{\$} \mathcal{X},\, y \xleftarrow{\$} \mathcal{Y},\, b' \leftarrow A^H(x, y)]$$
$$P_B = \Pr[x' = x : H \xleftarrow{\$} \Omega_H,\, x \xleftarrow{\$} \mathcal{X},\, x' \leftarrow C^H(x, i)]$$

Then, we have $|P_A^1 - P_A^2| \leq 2q\sqrt{P_B}$.

Proof (of Theorem 3.2). Let $\mathcal{A}$ be an adversary against the scheme Π_h above in the IND-CCA sense in the quantum random oracle model. Suppose $\mathcal{A}$ issues at most q_G, q_H, $q_{H'}$ and $q_{H''}$ quantum queries to the random oracles G, H, H' and H'' respectively and at most q_d classical decryption queries. Let Ω_G, Ω_H, $\Omega_{H'}$ and $\Omega_{H''}$ be the set of all functions $G : \{0,1\}^* \times \mathcal{M}_p \to \{0,1\}^{|G|}, H : \mathcal{M}_p \to \mathcal{K}_s$ and $H', H'' : \mathcal{M}_p \to \mathcal{M}_p$ respectively.

The game G_0 (see Fig. 2) is the IND-CCA security experiment associated to Π_h and $\mathcal{A}$. Hence, we have $\mathsf{Adv}^{\mathsf{ind-cca}}_{\Pi_h,\mathcal{A}}(t) = |2\Pr[\mathrm{G}_0 = 1] - 1|$. Now to bound the success probability of $\mathcal{A}$ in G_0, we introduce a sequence of *hybrid* games. Figure 2 describes games G_0 – G_3.

In game G_1, we introduce some "cosmetic" changes to the setup. Namely, we generate the values k^*, k^*, μ^* and c_1^* before $\mathcal{A}$ outputs the pair of challenge messages (m_0, m_1). This does not affect $\mathcal{A}$'s view in any way when it queries the oracles G, H, H', H'' and $\mathcal{D}_h$ in *phase one* (i.e., before $\mathcal{A}$ returns (m_0, m_1)). Hence, $\Pr[\mathrm{G}_1 = 1] = \Pr[\mathrm{G}_0 = 1]$.

In game G_2, we modify the decryption oracle $\mathcal{D}_h$ as follows: if $c_1 = c_1^*$, then we replace the hash evaluation "$\mu \leftarrow H'(k)$" with "$\mu \leftarrow \mu^*$". (We also make another "cosmetic change" to $\mathcal{D}_h$ where, if $k = \perp$, we make a *classical* H''-query on a uniformly random $k' \xleftarrow{\$} \mathcal{M}_p$. This change will become apparent later on when we make further modifications to $\mathcal{D}_h$ that allows us to decrypt any ciphertext without using sk.) Note that the games G_1 and G_2 are equivalent *unless* there is a decryption failure w.r.t. the ciphertext c_1^*. Since c_1^* is the encryption of a *random* message – i.e., $c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$, for $k^* \xleftarrow{\$} \mathcal{M}_p$ – we can bound the probability of such a failure event by δ. Therefore, $|\Pr[\mathrm{G}_1 = 1] - \Pr[\mathrm{G}_2 = 1]| \leq \delta$.

In the setup of game G_3, we replace the hash evaluations "$\mathsf{k}^* \leftarrow H(k^*)$" and "$\mu^* \leftarrow H'(k^*)$" with "$\mathsf{k}^* \xleftarrow{\$} \mathcal{K}_s$" and "$\mu^* \xleftarrow{\$} \mathcal{M}_p$" respectively. That is, k^* and μ^* are now uniformly random values that are generated independently of the (quantum) random oracles H and H' respectively. We first bound the success probability of $\mathcal{A}$ in G_3 via a reduction to the one-time IND-CPA security of Π_s. Let $\mathcal{B}$ be a one-time IND-CPA adversary against Π_s that, given an input 1^t, works as follows:

Games G_0 - G_3

$G \xleftarrow{\$} \Omega_G$, $H \xleftarrow{\$} \Omega_H$

$H' \xleftarrow{\$} \Omega_{H'}$, $H'' \xleftarrow{\$} \Omega_{H''}$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$

$k^* \xleftarrow{\$} \mathcal{M}_p$ /// G_1–G_3

$\mathsf{k}^* \leftarrow H(k^*)$, $\mu^* \leftarrow H'(k^*)$ /// G_1–G_2

$\mathsf{k}^* \xleftarrow{\$} \mathcal{K}_s$, $\mu^* \xleftarrow{\$} \mathcal{M}_p$ /// G_3

$c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$ /// G_1–G_3

$(m_0, m_1) \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$

$b \xleftarrow{\$} \{0,1\}$

$k^* \xleftarrow{\$} \mathcal{M}_p$ /// G_0

$\mathsf{k}^* \leftarrow H(k^*)$, $\mu^* \leftarrow H'(k^*)$ /// G_0

$r \xleftarrow{\$} \mathcal{R}_s$

$c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$ /// G_0

$c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r)$

$c_3^* \leftarrow G(c_2^*, \mu^*)$

$c_4^* \leftarrow H''(k^*)$

$b' \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$

return $(b' = b)$

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4))$

$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$

if $k = \perp$ **then**

  $k' \xleftarrow{\$} \mathcal{M}_p$, query $H''(k')$ /// G_2–G_3

  return $(\perp, \perp)$

$t \leftarrow H''(k)$

if $t \neq c_4$ **then return** $(\perp, \perp)$

if $c_1 = c_1^*$ **then** /// G_2–G_3

  $\mu \leftarrow \mu^*$, $t' \leftarrow G(c_2, \mu^*)$ /// G_2–G_3

else $\mu \leftarrow H'(k)$, $t' \leftarrow G(c_2, \mu)$

if $t' \neq c_3$ **then return** $(\perp, \perp)$

$\mathsf{k} \leftarrow H(k)$, $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$

return (k, m)

Fig. 2. Games G_0 – G_3 for the proof of Theorem 3.2.

- Runs $\mathcal{K}_h(1^t)$ to obtain $(\mathsf{pk}, \mathsf{sk})$.
- Generates $k^* \xleftarrow{\$} \mathcal{M}_p$, $\mu^* \xleftarrow{\$} \mathcal{M}_p$ and computes $c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$.
- Uses a $2(q_G + q_d + 1)$-wise independent function, $2(q_H + q_d + 1)$-wise independent function, $2(q_{H'} + q_d + 1)$-wise independent function and $2(q_{H''} + q_d + 1)$-wise independent function to simulate the quantum random oracles G, H, H' and H'' respectively, as noted in Lemma 3.1. (Considering the oracle G for example, the total number of times G is queried throughout the security experiment G_3 via the setup query "$c_3^* \leftarrow G(c_2^*, \mu^*)$", $\mathcal{A}$'s G-queries and decryption queries is at most $q_G + q_d + 1$.)
- Runs $\mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$ by answering the quantum random oracle queries and classical decryption queries as in G_3, and finally obtains (m_0, m_1).
- Forwards (m_0, m_1) to its one-time IND-CPA challenger and gets the ciphertext c_2^* in return. Note that the uniform secret key k^* is generated implicitly by the challenger (i.e., $\mathsf{k}^* \xleftarrow{\$} \mathcal{K}_s$) as well as the bit $b\,(\xleftarrow{\$} \{0,1\})$ and randomness $r\,(\xleftarrow{\$} \mathcal{R}_s)$. Thus, we have $c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r)$.
- Computes $c_3^* \leftarrow G(c_2^*, \mu^*)$ and $c_4^* \leftarrow H''(k^*)$.
- Runs $\mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$ by answering the random oracle queries and decryption queries as in G_3, and finally obtains a bit b'.
- Forwards bit b' to its (one-time) IND-CPA challenger as the final message.

$A^{H\times H'}(k^*, (\mathsf{k}^*, \mu^*))$

1 : $G \xleftarrow{\$} \Omega_G, H'' \xleftarrow{\$} \Omega_{H''}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$
3 : $c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$
4 : $(m_0, m_1) \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$
5 : $b \xleftarrow{\$} \{0,1\}$, $r \xleftarrow{\$} \mathcal{R}_s$
6 : $c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r)$
7 : $c_3^* \leftarrow G(c_2^*, \mu^*)$
8 : $c_4^* \leftarrow H''(k^*)$
9 : $b' \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$
10 : **return** $(b' = b)$

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4))$

1 : $k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$
2 : **if** $k = \perp$ **then**
3 : $\quad k' \xleftarrow{\$} \mathcal{M}_p$, query $H''(k')$
4 : $\quad$ **return** $(\perp, \perp)$
5 : $t \leftarrow H''(k)$
6 : **if** $t \neq c_4$ **then return** $(\perp, \perp)$
7 : **if** $c_1 = c_1^*$ **then**
8 : $\quad \mu \leftarrow \mu^*$, $t' \leftarrow G(c_2, \mu^*)$
9 : **else** $\mu \leftarrow H'(k)$, $t' \leftarrow G(c_2, \mu)$
10 : **if** $t' \neq c_3$ **then return** $(\perp, \perp)$
11 : $\mathsf{k} \leftarrow H(k)$, $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
12 : **return** (k, m)

Fig. 3. Algorithm $A^{H\times H'}$ for the proof of Theorem 3.2.

It is easy to see that $\mathsf{Adv}^{\mathsf{ind-cpa}}_{\Pi_s,\mathcal{B}}(t) = |2\Pr[\mathrm{G}_3 = 1] - 1|$. Now using Lemma 3.3, we bound the difference between the success probabilities of $\mathcal{A}$ in G_2 and G_3. Let A be an algorithm that has quantum access to the random oracle $H \times H'$, where $(H \times H')(k) = (H(k), H'(k))$. Figure 3 describes $A^{H\times H'}$'s operation on input $(k^*, (\mathsf{k}^*, \mu^*))$. Note that the algorithm $A^{H\times H'}$ makes at most $q_H + q_{H'} + 2q_d$ number of queries to the random oracle $H \times H'$ to respond to $\mathcal{A}$'s oracle queries[11].

[11] For example, if $A^{H\times H'}$ wants to respond to $\mathcal{A}$'s H-query, then $A^{H\times H'}$ prepares a uniform superposition of all states in the output register corresponding to H' (see [TU16] for particulars of this "trick").

Games G_4 - G_7, G_9, G_{11}

$G \xleftarrow{\$} \Omega_G, H \xleftarrow{\$} \Omega_H, H' \xleftarrow{\$} \Omega_{H'}$

$H'' \xleftarrow{\$} \Omega_{H''}$ // G_4 – G_7

$H'' \xleftarrow{\$} \Omega_{poly}$ // G_9, G_{11}

$R \xleftarrow{\$} \Omega_R$ // G_5

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$

$k^* \xleftarrow{\$} \mathcal{M}_p$, $\mathsf{k}^* \xleftarrow{\$} \mathcal{K}_s$, $\mu^* \xleftarrow{\$} \mathcal{M}_p$

$c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$

$i \xleftarrow{\$} \{1, \ldots, q_1\}$

run until i-th query to oracle $H \times H'$

  $(m_0, m_1) \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$

  $b \xleftarrow{\$} \{0,1\}$, $r \xleftarrow{\$} \mathcal{R}_s$

  $c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r)$

  $c_3^* \leftarrow G(c_2^*, \mu^*)$ // G_4

  $c_3^* \leftarrow R(c_2^*)$ // G_5

  $c_3^* \xleftarrow{\$} \{0,1\}^{|G|}$ // G_6–G_7, G_9, G_{11}

  $c_4^* \leftarrow H''(k^*)$ // G_4–G_6

  $c_4^* \xleftarrow{\$} \mathcal{M}_p$ // G_7, G_9, G_{11}

  $b' \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$

measure the argument $\hat{k}$ of the i-th query to oracle $H \times H'$

return $(\hat{k} = k^*)$

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4))$ // G_4 - G_7, G_9

if $c_1 = c_1^*$ **then**

  return $(\perp, \perp)$ // G_6–G_7, G_9

$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$

if $k = \perp$ **then**

  $k' \xleftarrow{\$} \mathcal{M}_p$, query $H''(k')$

  return $(\perp, \perp)$

$t \leftarrow H''(k)$

if $t \neq c_4$ **then return** $(\perp, \perp)$

if $c_1 = c_1^*$ **then** // G_4–G_5

  $\mu \leftarrow \mu^*$, $t' \leftarrow G(c_2, \mu^*)$ // G_4

  $t' \leftarrow R(c_2)$ // G_5

else $\mu \leftarrow H'(k)$, $t' \leftarrow G(c_2, \mu)$

if $t' \neq c_3$ **then return** $(\perp, \perp)$

$\mathsf{k} \leftarrow H(k)$, $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$

return (k, m)

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4))$ // G_{11}

if $c_1 = c_1^*$ **then**

  return $(\perp, \perp)$

Compute set of roots S

  of polynomial $H''(x) - c_4$

if $\exists k \in S$ s.t. $\mathcal{E}_p(\mathsf{pk}, k) = c_1$

then

  query $H''(k)$

  $\mu \leftarrow H'(k)$, $t' \leftarrow G(c_2, \mu)$

  if $t' \neq c_3$ **then**

    return $(\perp, \perp)$

  else $\mathsf{k} \leftarrow H(k)$, $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$

    return (k, m)

else $k' \xleftarrow{\$} \mathcal{M}_p$, query $H''(k')$

  return $(\perp, \perp)$

Fig. 4. Games G_4 – G_7, G_9, G_{11} for the proof of Theorem 3.2.

Let $q_1 = q_H + q_{H'} + 2q_d$ and let B be an oracle algorithm that on input k^* does the following: picks $i \xleftarrow{\$} \{1, \ldots, q_1\}$, generates $\mathsf{k}^* \xleftarrow{\$} \mathcal{K}_s$ and $\mu^* \xleftarrow{\$} \mathcal{M}_p$, runs $A^{H \times H'}(k^*, (\mathsf{k}^*, \mu^*))$ until the i-th query, measures the argument of the $(H \times H')$-query in the computational basis and outputs the measurement outcome (if $A^{H \times H'}$ makes less than i queries, B outputs $\perp$). With this construction of A, note that $P_A^1 = \Pr[\mathrm{G}_2 = 1]$ and $P_A^2 = \Pr[\mathrm{G}_3 = 1]$, where P_A^1 and P_A^2 are as defined in Lemma 3.3 w.r.t. the algorithm $A^{H \times H'}$. Therefore, we now define game G_4 (see Fig. 4) such that $P_B = \Pr[\mathrm{G}_4 = 1]$, where P_B is as defined in Lemma 3.3 w.r.t. the algorithm $B^{H \times H'}$. From Lemma 3.3, we thus have

$$|\Pr[\mathrm{G}_2 = 1] - \Pr[\mathrm{G}_3 = 1]| \;\leq\; 2(q_H + q_{H'} + 2q_d)\sqrt{\Pr[\mathrm{G}_4 = 1]}$$

Now Figure 4 describes games G_4 – G_7, G_9 and G_{11}. From Lemma 3.2, in game G_5, we replace the (quantum-accessible) "pseudorandom function" $G(.\,, \mu^*)$ with a (quantum) "truly" random oracle $R(.)$. Specifically, let Ω_R be the set of all functions $R : \{0,1\}^* \to \{0,1\}^{|G|}$. Then $R(\xleftarrow{\$} \Omega_R)$ is an internal oracle that is not directly accessible by $\mathcal{A}$. We justify this replacement using Lemma 3.2 w.r.t. a distinguisher C^{G,F_j} ($j \in \{0,1\}$) described in Figure 5. Here $F_0(.) = G(.\,, \mu^*)$ for $\mu^* \xleftarrow{\$} \mathcal{M}_p$, and $F_1(.) = R(.)$. It is not hard to see that C^{G,F_0} (respectively, C^{G,F_1}) perfectly simulates G_4 (respectively, G_5) towards $\mathcal{A}$. Since the uniform secret "PRF key" μ^* is generated independently from C^{G,F_j}'s view, we use Lemma 3.2 to obtain the following (note that C makes at most $q_G + q_d + 1$ queries to the oracles G and F_j)

$$|\Pr[\mathrm{G}_4 = 1] - \Pr[\mathrm{G}_5 = 1]| \;\leq\; \frac{2(q_G + q_d + 1)}{\sqrt{|\mathcal{M}_p|}}$$

In game G_6, we modify the decryption oracle as follows: if $c_1 = c_1^*$, return $(\perp, \perp)$. (We also make a "cosmetic" change where we replace "$c_3^* \leftarrow R(c_2^*)$" with "$c_3^* \xleftarrow{\$} \{0,1\}^{|G|}$", since the random function $R(.)$ would have *only* been used on c_2^* throughout G_6 and no other c_2-values.) Note that the only way the execution of games G_5 and G_6 would differ is if $\mathcal{A}$ made decryption queries of the form (c_1^*, c_2, c_3, c_4^*) where $c_2 \neq c_2^*$ and $R(c_2) = c_3$; also in such an event, the number of H-queries with argument k^* in G_5 and G_6 will go "out of sync" resulting in a difference in $\mathcal{A}$'s respective success probabilities. Since R is an internal random function not directly accessible by $\mathcal{A}$, we can bound the probability of the event "$R(c_2) = c_3$" w.r.t. a *single* decryption query (c_1^*, c_2, c_3, c_4^*) by $1/2^{|G|}$. Using a union bound, we conclude that

$$|\Pr[\mathrm{G}_5 = 1] - \Pr[\mathrm{G}_6 = 1]| \;\leq\; \frac{q_d}{2^{|G|}}$$

In the setup of game G_7, we replace the computation "$c_4^* \leftarrow H''(k^*)$" with "$c_4^* \xleftarrow{\$} \mathcal{M}_p$". That is, c_4^* is now a uniformly random value that is generated independently of k^* and the (quantum) random oracle H''. Using Lemma 3.3, we bound the difference between the success probabilities of $\mathcal{A}$ in G_6 and G_7. Let $\hat{A}$ be an algorithm that has quantum access to the random oracle H''. Figure 6 describes $\hat{A}^{H''}$'s operation on input (k^*, c_4^*). Note that the algorithm $\hat{A}^{H''}$ makes at most $q_{H''} + q_d$ queries to the random oracle H'' to respond to $\mathcal{A}$'s oracle queries.

Let $q_2 = q_{H''} + q_d$ and let $\hat{B}$ be an oracle algorithm that on input k^* does the following: picks $j \xleftarrow{\$} \{1, \ldots, q_2\}$, generates $c_4^* \xleftarrow{\$} \mathcal{M}_p$, runs $\hat{A}^{H''}(k^*, c_4^*)$ until the j-th query, measures the argument of the H''-query in the computational basis and outputs the measurement outcome (if $\hat{A}^{H''}$ makes

C^{G,F_j}

$H \xleftarrow{\$} \Omega_H, H' \xleftarrow{\$} \Omega_{H'}, H'' \xleftarrow{\$} \Omega_{H''}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$
$k^* \xleftarrow{\$} \mathcal{M}_p, \mathsf{k}^* \xleftarrow{\$} \mathcal{M}_p$
$c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$
$i \xleftarrow{\$} \{1, \ldots, q_1\}$
run until i-th query to oracle $H \times H'$
  $(m_0, m_1) \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$
  $b \xleftarrow{\$} \{0,1\}$, $r \xleftarrow{\$} \mathcal{R}_s$
  $c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r)$
  $c_3^* \leftarrow F_j(c_2^*)$
  $c_4^* \leftarrow H''(k^*)$
  $b' \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$
measure the argument $\hat{k}$ of the i-th query to oracle $H \times H'$
return $(\hat{k} = k^*)$

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4))$

$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$
if $k = \perp$ **then**
  $k' \xleftarrow{\$} \mathcal{M}_p$, query $H''(k')$
  return $(\perp, \perp)$
$t \leftarrow H''(k)$
if $t \neq c_4$ **then return** $(\perp, \perp)$
if $c_1 = c_1^*$ **then** $t' \leftarrow F_j(c_2)$
else $\mu \leftarrow H'(k)$, $t' \leftarrow G(c_2, \mu)$
if $t' \neq c_3$ **then return** $(\perp, \perp)$
$\mathsf{k} \leftarrow H(k)$, $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
return (k, m)

Fig. 5. Algorithm C^{G,F_i} for the proof of Theorem 3.2.

$\hat{A}^{H''}(k^*, c_4^*)$

$G \xleftarrow{\$} \Omega_G, H \xleftarrow{\$} \Omega_H, H' \xleftarrow{\$} \Omega_{H'}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$
$\mathsf{k}^* \xleftarrow{\$} \mathcal{M}_p$
$c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$
$i \xleftarrow{\$} \{1, \ldots, q_1\}$
run until i-th query to oracle $H \times H'$
  $(m_0, m_1) \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$
  $b \xleftarrow{\$} \{0,1\}$, $r \xleftarrow{\$} \mathcal{R}_s$
  $c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r)$
  $c_3^* \xleftarrow{\$} \{0,1\}^{|G|}$
  $b' \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$
measure the argument $\hat{k}$ of the i-th query to oracle $H \times H'$
return $(\hat{k} = k^*)$

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4))$

if $c_1 = c_1^*$ **then return** $(\perp, \perp)$
$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$
if $k = \perp$ **then**
  $k' \xleftarrow{\$} \mathcal{M}_p$, query $H''(k')$
  return $(\perp, \perp)$
$t \leftarrow H''(k)$
if $t \neq c_4$ **then return** $(\perp, \perp)$
$\mu \leftarrow H'(k)$, $t' \leftarrow G(c_2, \mu)$
if $t' \neq c_3$ **then return** $(\perp, \perp)$
$\mathsf{k} \leftarrow H(k)$, $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
return (k, m)

Fig. 6. Algorithm $\hat{A}^{H''}$ for the proof of Theorem 3.2.

Games G_8 and G_{10}

$G \xleftarrow{\$} \Omega_G, H \xleftarrow{\$} \Omega_H, H' \xleftarrow{\$} \Omega_{H'}$

$H'' \xleftarrow{\$} \Omega_{H''} /\!\!/ G_8$

$H'' \xleftarrow{\$} \Omega_{poly} /\!\!/ G_{10}, G_{12}$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}_p(1^t)$

$k^* \xleftarrow{\$} \mathcal{M}_p$, $\mathsf{k}^* \xleftarrow{\$} \mathcal{K}_s$, $\mu^* \xleftarrow{\$} \mathcal{M}_p$

$c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$

$j \xleftarrow{\$} \{1, \ldots, q_2\}$

run until j-th query to oracle H''

  $i \xleftarrow{\$} \{1, \ldots, q_1\}$

  run until i-th query to oracle $H \times H'$

    $(m_0, m_1) \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$

    $b \xleftarrow{\$} \{0,1\}$, $r \xleftarrow{\$} \mathcal{R}_s$

    $c_2^* \leftarrow \mathcal{E}_s(\mathsf{k}^*, m_b; r)$

    $c_3^* \xleftarrow{\$} \{0,1\}^{|G|}$

    $c_4^* \xleftarrow{\$} \mathcal{M}_p$

    $b' \leftarrow \mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$

  measure the argument $\hat{k}$ of the i-th query to oracle $H \times H'$

measure the argument $\tilde{k}$ of the j-th query to oracle H''

return $(\tilde{k} = k^*)$

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4)) /\!\!/ G_8, G_{10}$

if $c_1 = c_1^*$ **then**
  return $(\perp, \perp)$
$k \leftarrow \mathcal{D}_p(\mathsf{sk}, c_1)$
if $k = \perp$ **then**
  $k' \xleftarrow{\$} \mathcal{M}_p$, query $H''(k')$
  return $(\perp, \perp)$
$t \leftarrow H''(k)$
if $t \neq c_4$ **then return** $(\perp, \perp)$
else $\mu \leftarrow H'(k)$, $t' \leftarrow G(c_2, \mu)$
if $t' \neq c_3$ **then return** $(\perp, \perp)$
$\mathsf{k} \leftarrow H(k)$, $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
return (k, m)

$\mathcal{D}_h(\mathsf{sk}, (c_1, c_2, c_3, c_4)) /\!\!/ G_{12}$

if $c_1 = c_1^*$ **then**
  return $(\perp, \perp)$
Compute set of roots S
  of polynomial $H''(x) - c_4$
if $\exists k \in S$ s.t. $\mathcal{E}_p(\mathsf{pk}, k) = c_1$
then
  query $H''(k)$
  $\mu \leftarrow H'(k)$, $t' \leftarrow G(c_2, \mu)$
  if $t' \neq c_3$ **then**
    return $(\perp, \perp)$
  else $\mathsf{k} \leftarrow H(k)$, $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
    return (k, m)
else $k' \xleftarrow{\$} \mathcal{M}_p$, query $H''(k')$
  return $(\perp, \perp)$

Fig. 7. Games G_8, G_{10} and G_{12} for the proof of Theorem 3.2.

less than j queries, $\hat{B}$ outputs $\perp$). With this construction of $\hat{A}$, note that $P^1_{\hat{A}} = \Pr[\mathrm{G}_6 = 1]$ and $P^2_{\hat{A}} = \Pr[\mathrm{G}_7 = 1]$, where $P^1_{\hat{A}}$ and $P^2_{\hat{A}}$ are as defined in Lemma 3.3 w.r.t. the algorithm $\hat{A}^{H''}$. Therefore, we now define game G_8 (see Figure 7 which describes games G_8, G_{10} and G_{12}) such that $P_{\hat{B}} = \Pr[\mathrm{G}_8 = 1]$, where $P_{\hat{B}}$ is as defined in Lemma 3.3 w.r.t. the algorithm $\hat{B}^{H''}$. From Lemma 3.3, we thus have

$$|\Pr[\mathrm{G}_6 = 1] - \Pr[\mathrm{G}_7 = 1]| \;\leq\; 2(q_{H''} + q_d)\sqrt{\Pr[\mathrm{G}_8 = 1]}$$

In games G_9 and G_{10}, we replace the random oracle H'' with a $2(q_{H''}+q_d+1)$-wise independent function, following Lemma 3.1. Random polynomials of degree $2(q_{H''} + q_d + 1) - 1$ over the finite field representation of $\mathcal{M}_p$ are $2(q_{H''} + q_d + 1)$-wise independent. Let Ω_{poly} be the set of all such polynomials. Then specifically, we are replacing the step "$H'' \xleftarrow{\$} \Omega_{H''}$" with "$H'' \xleftarrow{\$} \Omega_{poly}$" in both games. From Lemma 3.1, as this change is indistinguishable when the oracle H'' is queried at most $q_{H''} + q_d + 1$ times, we have G_7 and G_9 (respectively, G_8 and G_{10}) to be equivalent. Therefore, $\Pr[\mathrm{G}_7 = 1] = \Pr[\mathrm{G}_9 = 1]$ and $\Pr[\mathrm{G}_8 = 1] = \Pr[\mathrm{G}_{10} = 1]$.

In G_{11} and G_{12}, we modify the decryption oracle – the same way in both games (Fig. 4 describes G_{11} and Fig. 7 describes G_{12}, respectively) – such that the secret key sk is not used to decrypt a ciphertext (c_1, c_2, c_3, c_4). To analyze this change to $\mathcal{D}_h$, we define two "bad" events in games G_9 – G_{12}:

- Let bad_1 denote the event that $\mathcal{A}$ asks for the decryption of (c_1, c_2, c_3, c_4) such that c_1 is a ciphertext for which there are two *distinct* messages k, k' that encrypt to it – i.e., $\mathcal{E}_p(\mathsf{pk}, k) = \mathcal{E}_p(\mathsf{pk}, k') = c_1$.
- Let bad_2 denote the event that $\mathcal{A}$ asks for the decryption of (c_1, c_2, c_3, c_4) such that $\mathcal{D}_p(\mathsf{sk}, c_1) = \perp$ *and* there exists a root k' of the polynomial $H''(x) - c_4$ (recall that H'' in now a random *polynomial* of degree $(2q_{H''} + q_d + 1) - 1$) which satisfies $\mathcal{E}_p(\mathsf{pk}, k') = c_1$.

Setting $\mathsf{bad} = \mathsf{bad}_1 \vee \mathsf{bad}_2$, we have the following w.r.t. games G_9 and G_{10}:

$$\Pr[\mathrm{G}_9 = 1] \leq \Pr[\mathsf{bad}] + \Pr[\neg\mathsf{bad}]\Pr[\mathrm{G}_9 = 1 \mid \neg\mathsf{bad}]$$
$$\Pr[\mathrm{G}_{10} = 1] \leq \Pr[\mathsf{bad}] + \Pr[\neg\mathsf{bad}]\Pr[\mathrm{G}_{10} = 1 \mid \neg\mathsf{bad}]$$

Now to show $\Pr[\mathrm{G}_9 = 1|\neg\mathsf{bad}] \leq \Pr[\mathrm{G}_{11} = 1|\neg\mathsf{bad}]$ and $\Pr[\mathrm{G}_{10} = 1|\neg\mathsf{bad}] \leq \Pr[\mathrm{G}_{12} = 1|\neg\mathsf{bad}]$, it is sufficient to show that, assuming the event $\neg\mathsf{bad}$ occurs: (1) the new decryption oracle returns the same output as the previous oracle when queried on any ciphertext, (2) the queries submitted to the random oracles H, H' and H'' remain "in sync" after this modification to $\mathcal{D}_h$ (e.g., the j-th query to H'' at a particular stage in G_{10} corresponds to the j-th query to H'' in the *same* stage of G_{12}), and (3) upon measuring the argument of the i-th query to oracle $(H \times H')$ in G_9 and G_{11} (resp., the j-th query to oracle H'' in G_{10} and G_{12}), the probability of the outcome being k^* in G_{11} (resp., G_{12}) is greater than or equal to that in G_9 (resp., G_{10}).

Suppose $\mathcal{A}$ asks for the decryption of (c_1, c_2, c_3, c_4). Let $k = \mathcal{D}_p(\mathsf{sk}, c_1)$. Consider the following cases while assuming the event $\neg\mathsf{bad}$ occurs::

1. If $c_1 = c_1^*$, then the $\mathcal{D}_h$ oracle in G_9, G_{10}, G_{11} and G_{12} returns $(\perp, \perp)$. At the same time, no queries are made to H, H' and H'' at this stage (in particular, no query on k^*).
2. If $c_1 \neq c_1^*$ and $k = \perp$, then the oracle $\mathcal{D}_h$ in G_9 and G_{10} returns $(\perp, \perp)$. Since the event $\neg\mathsf{bad}$ (and hence, $\neg\mathsf{bad}_2$) happens, the oracle $\mathcal{D}_h$ in G_{11} and G_{12} returns $(\perp, \perp)$ as well, as there does not exist a root $k' \in S$ such that $\mathcal{E}_p(\mathsf{pk}, k') = c_1$.

No queries are made to H and H' in this case. On the other hand, a single *classical* query is made to H'' on a uniformly random value in $\mathcal{M}_p$ in games G_9 – G_{12}. Hence in particular, the probability of the query being k^* is equal in G_{10} and G_{12}.

3. If $c_1 \neq c_1^*$, $k \neq \perp$ and $H''(k) \neq c_4$, then $\mathcal{D}_h$ in G_9 and G_{10} returns $(\perp, \perp)$. Since the event $\neg\mathsf{bad}$ (and hence, $\neg\mathsf{bad}_1$) happens, $\mathcal{D}_h$ in G_{11} and G_{12} returns $(\perp, \perp)$ as well, as there does not exist a root $k' \in S$ such that $\mathcal{E}_p(\mathsf{pk}, k') = c_1$; otherwise, from the rigidity of Π_p, we have $\mathcal{E}_p(\mathsf{pk}, k) = c_1 = \mathcal{E}_p(\mathsf{pk}, k')$ with $k \neq k'$ (since $H''(k) \neq c_4 = H''(k')$), contradicting the event $\neg\mathsf{bad}_1$ happening.

   No queries are made to H and H' in this case. In games G_9, G_{10}, a *classical* query is made to H'' on k, to do the check "$(H''(k) = c_4)$". As Π_p is rigid, we have $\mathcal{E}_p(\mathsf{pk}, k) = c_1 \neq c_1^*$. Since Π_p is also deterministic, it must be the case that $k \neq k^*$. In G_{11} and G_{12}, as there does not exist a root $k' \in S$ such that $\mathcal{E}_p(\mathsf{pk}, k') = c_1$, we make a *classical* H''-query on a uniformly random value from $\mathcal{M}_p$. This step essentially keeps the H''-oracle calls "in sync" across both decryption oracles. Now it is not hard to see that if the j-th query to oracle H'' – where $j \xleftarrow{\$} \{1, \ldots, q_2\}$ was sampled at the beginning of G_{10} and G_{12} – is at this stage, namely when $\mathcal{A}$ made this particular decryption query, then the probability of the measurement outcome w.r.t. this *classical* H''-query being k^* is 0 in G_{10} and $1/|\mathcal{M}_p|$ in G_{12}.

4. If $c_1 \neq c_1^*$, $k \neq \perp$, $H''(k) = c_4$ and $G(c_2, H'(k)) \neq c_3$, then $\mathcal{D}_h$ in G_9 and G_{10} returns $(\perp, \perp)$. $\mathcal{D}_h$ in G_{11} and G_{12} also returns $(\perp, \perp)$, as now we have $k \in S$ (since $H''(k) - c_4 = 0$) such that $\mathcal{E}_p(\mathsf{pk}, k) = c_1$, which follows from Π_p's rigidity. At the same time, as the event $\neg\mathsf{bad}$ (and hence, $\neg\mathsf{bad}_1$) happens, there must not exist a *different* root $k' \in S$ such that $\mathcal{E}_p(\mathsf{pk}, k') = c_1$. Since the G-check fails w.r.t. k in this new decryption oracle as well, i.e., $G(c_2, H'(k)) \neq c_3$, $(\perp, \perp)$ is returned.

   No query is made to H in this case. But a *classical* query is made to H' and H'' on the value k at this stage in G_9 – G_{12}. Thus, all oracles call are "in sync" across both versions of $\mathcal{D}_h$, and the probability of the measurement outcome w.r.t. this *classical* $(H \times H')$-query (resp., H''-query) in G_9 and G_{11} (resp., G_{10} and G_{12}) being k^* is 0.

5. If $c_1 \neq c_1^*$, $k \neq \perp$, $H''(k) = c_4$ and $G(c_2, H'(k)) = c_3$, then $\mathcal{D}_h$ in G_9 and G_{10} returns $(k, \mathcal{D}_s(H(k), c_2))$. $\mathcal{D}_h$ in G_{11} and G_{12} also returns $(k, \mathcal{D}_s(H(k), c_2))$ following a similar analysis above, the only difference being that now the (sole) root in S, namely k, also satisfies the G-check: $G(c_2, H'(k)) = c_3$.

   In this case, a *classical* query is made to H, H' and H'' on k in G_9 – G_{12}. Again, all oracles call are "in sync" across both versions of $\mathcal{D}_h$, and the probability of the measurement outcome w.r.t. any of the two *classical* $(H \times H')$-queries, corresponding to the $H(k)$ and $H'(k)$ calls respectively, in G_9 and G_{11} (resp., the single H''-query, corresponding to the $H''(k)$ call, in G_{10} and G_{12}) being k^* is 0.

Thus, we have $\Pr[\mathrm{G}_9 = 1 | \neg\mathsf{bad}] \leq \Pr[\mathrm{G}_{11} = 1 | \neg\mathsf{bad}]$ and $\Pr[\mathrm{G}_{10} = 1 | \neg\mathsf{bad}] \leq \Pr[\mathrm{G}_{12} = 1 | \neg\mathsf{bad}]$. At the same time, it is not hard to see that the probability $\Pr[\neg\mathsf{bad}]$ (and hence, $\Pr[\mathsf{bad}]$) should be the same in games G_9 and G_{11} (resp., G_{10} and G_{12}). This is because, the event $\neg\mathsf{bad}$ depends on $\mathcal{A}$'s queries to the $\mathcal{D}_h$ oracle. In the above analysis, since we showed that – assuming the event $\neg\mathsf{bad}$ occurs – the $\mathcal{D}_h$ oracles have the same input-output behavior in G_9 – G_{12}, $\mathcal{A}$'s view (and hence, execution) in G_9 and G_{11} (resp., G_{10} and G_{12}) should be *identical* until the first $\mathcal{D}_h$-query that violates $\neg\mathsf{bad}$; this means the probability that a given $\mathcal{D}_h$-query made by $\mathcal{A}$ satisfies the event $\neg\mathsf{bad}$ remains the same in G_9 and G_{11} (resp., G_{10} and G_{12}) while conditioning on the event that

all of $\mathcal{A}$'s previous $\mathcal{D}_h$-queries are consistent with $\neg\mathsf{bad}$. So we first have the following:

$$\Pr[\mathrm{G}_9 = 1] \leq \Pr[\mathsf{bad}] + \Pr[\neg\mathsf{bad} \wedge \mathrm{G}_{11} = 1] \leq \Pr[\mathsf{bad}] + \Pr[\mathrm{G}_{11} = 1]$$
$$\Pr[\mathrm{G}_{10} = 1] \leq \Pr[\mathsf{bad}] + \Pr[\neg\mathsf{bad} \wedge \mathrm{G}_{12} = 1] \leq \Pr[\mathsf{bad}] + \Pr[\mathrm{G}_{12} = 1]$$

Now we bound $\Pr[\mathsf{bad}]$ by analyzing the event "bad" in games G_{11} and G_{12}. Recall that $\mathsf{bad} = \mathsf{bad}_1 \vee \mathsf{bad}_2$, and hence by a union bound, we have $\Pr[\mathsf{bad}] \leq \Pr[\mathsf{bad}_1] + \Pr[\mathsf{bad}_2]$. Since Π_p is δ_c-Collision Free, we have $\Pr[\mathsf{bad}_1] \leq \delta_c$. We bound the term $\Pr[\mathsf{bad}_2]$ via a reduction to the $\perp$-Aware security of Π_p. Let $\mathcal{C}$ be a $\perp$-Aware adversary against Π_p that, given an input pk, works as follows:

- Generates $k^* \xleftarrow{\$} \mathcal{M}_p$, $\mathsf{k}^* \xleftarrow{\$} \mathcal{K}_s$, $\mu^* \xleftarrow{\$} \mathcal{M}_p$ and sets $c_1^* \leftarrow \mathcal{E}_p(\mathsf{pk}, k^*)$.
- Uses a $2(q_G + q_d + 1)$-wise independent function, $2(q_H + q_d + 1)$-wise independent function, $2(q_{H'} + q_d + 1)$-wise independent function and $2(q_{H''} + q_d + 1)$-wise independent polynomial to simulate the quantum random oracles G, H, H' and H'' respectively, as noted in Lemma 3.1.
- Initializes an empty list $\mathcal{L}$.
- Runs $\mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$ by answering the quantum random oracle queries and classical decryption queries as in G_{11}, and finally obtains (m_0, m_1). At the same time, stores each of $\mathcal{A}$'s *classical* decryption queries in $\mathcal{L}$.
  (Note that $\mathcal{C}$ can simulate $\mathcal{D}_h$ of G_{11} without having access to the decryption oracle $\mathcal{D}_p(\mathsf{sk}, .)$.)
- Samples a bit $b \xleftarrow{\$} \{0,1\}$ and the random coins $r \in \mathcal{R}_s$ for the symmetric Π_s-encryption to compute $c_2^* = \mathcal{E}_s(\mathsf{k}^*, m_b; r)$. Generates the rest of the ciphertext components as $c_3^* \xleftarrow{\$} \{0,1\}^{|G|}$ and $c_4^* \xleftarrow{\$} \mathcal{M}_p$.
- Runs $\mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$ by answering the quantum random oracle queries and classical decryption queries as in G_{11}, and finally obtains a bit b'. Again stores each of $\mathcal{A}$'s *classical* decryption queries in $\mathcal{L}$.
- Selects a ciphertext (c_1, c_2, c_3, c_4) uniformly *at random* from the list $\mathcal{L}$ and does the following:
  - Computes set of roots S of polynomial $H''(x) - c_4$ and checks if there exists a root $k' \in S$ such that $\mathcal{E}_p(\mathsf{pk}, k') = c_1$. (Note that finding roots of polynomial $H''(x) - c_4$ is polynomial-time computable.)
  - If there exists such a root $k' \in S$, returns the pair (k', c_1) to its $\perp$-Aware challenger and halts. Otherwise, returns the pair $(\perp, \perp)$ to its $\perp$-Aware challenger and halts.

It is easy to see that if the event bad_2 occurs in game G_{11}, then with probability at least $1/q_d$, $\mathcal{C}$ will select the *right* ciphertext (c_1, c_2, c_3, c_4) from the list $\mathcal{L}$ that results in a $\perp$-Aware game-winning pair (k', c_1). Thus, we have $\Pr[\mathsf{bad}_2] \leq q_d \cdot \mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi_p,\mathcal{C}}(t)$. A similar analysis will also hold for G_{12}, where $\Pr[\mathsf{bad}_1] \leq \delta_c$ and $\Pr[\mathsf{bad}_2] \leq q_d \cdot \mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi_p,\mathcal{C}}(t)$ for the same reduction to $\perp$-Aware security of Π_p as in G_{11} – the reason being that the $\mathcal{D}_h$ oracles are the same in games G_{11} and G_{12}. Therefore, we get

$$\Pr[\mathrm{G}_9 = 1] \leq \delta_c + q_d \cdot \mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi_p,\mathcal{C}}(t) + \Pr[\mathrm{G}_{11} = 1]$$
$$\Pr[\mathrm{G}_{10} = 1] \leq \delta_c + q_d \cdot \mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi_p,\mathcal{C}}(t) + \Pr[\mathrm{G}_{12} = 1]$$

Now since the $\mathcal{D}_h$ oracle in games G_{11} and G_{12} does not use the secret key sk to decrypt any ciphertext, we can bound the success probability of $\mathcal{A}$ in G_{11} and G_{12} via reductions to the OW-CPA security of Π_p. Let $\mathcal{D}$ (resp., $\mathcal{E}$) be an OW-CPA adversary against Π_p that, given an input $(1^t, \mathsf{pk}, c^*)$, works as follows:

- Generates $\mathsf{k}^* \xleftarrow{\$} \mathcal{K}_s$, $\mu^* \xleftarrow{\$} \mathcal{M}_p$ and sets $c_1^* \leftarrow c^*$. Note that the uniform message k^* is generated implicitly by the OW-CPA challenger (along with the public key pk) such that $\mathcal{E}_p(\mathsf{pk}, k^*) = c_1^*$ $(= c^*)$.
- Uses a $2(q_G + q_d + 1)$-wise independent function, $2(q_H + q_d + 1)$-wise independent function, $2(q_{H'} + q_d + 1)$-wise independent function and $2(q_{H''} + q_d + 1)$-wise independent polynomial to simulate the quantum random oracles G, H, H' and H'' respectively, as noted in Lemma 3.1.
- Selects $i \xleftarrow{\$} \{1, \ldots, q_1\}$ (resp., $j \xleftarrow{\$} \{1, \ldots, q_2\}$).
- Until the i-th (resp., j-th) query to the oracle $H \times H'$ (resp., H'') is made, does the following:
  - Runs $\mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(\mathsf{pk})$ by answering the quantum random oracle queries and classical decryption queries as in G_{11} (resp., G_{12}), and finally obtains (m_0, m_1).
    (Note that the OW-CPA adversaries $\mathcal{D}$ and $\mathcal{E}$ can simulate the decryption oracle $\mathcal{D}_h$ without possessing the secret key sk.)
  - Samples a bit $b \xleftarrow{\$} \{0,1\}$ and the random coins $r \in \mathcal{R}_s$ for the symmetric Π_s-encryption to compute $c_2^* = \mathcal{E}_s(\mathsf{k}^*, m_b; r)$. Generates the rest of the ciphertext components as $c_3^* \xleftarrow{\$} \{0,1\}^{|G|}$ and $c_4^* \xleftarrow{\$} \mathcal{M}_p$.
  - Runs $\mathcal{A}^{G,H,H',H'',\mathcal{D}_h}(c_1^*, c_2^*, c_3^*, c_4^*)$ by answering the quantum random oracle queries and classical decryption queries as in G_{11} (resp., G_{12}), and obtains a bit b'.
- Measures the argument $\hat{k}$ of the i-th (resp., j-th) query to the oracle $H \times H'$ (resp., H'') and outputs $\hat{k}$; if $\mathcal{A}$ makes less than i (resp., j) queries, output $\perp$.

From the above construction of adversaries $\mathcal{D}$ and $\mathcal{E}$, it is easy to see that $\Pr[\mathrm{G}_{11} = 1] \leq \frac{1}{|\mathcal{M}_p|} + \mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi_p,\mathcal{D}}(t)$ and $\Pr[\mathrm{G}_{12} = 1] \leq \frac{1}{|\mathcal{M}_p|} + \mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi_p,\mathcal{E}}(t)$.

Setting $q_3 = 2(q_G + q_d + 1)$ and $\delta' = \delta_c + q_d \cdot \mathsf{Adv}^{\perp\text{-}\mathsf{Aware}}_{\Pi_p,\mathcal{C}}(t) + 1/|\mathcal{M}_p|$, by combining all the above bounds w.r.t. the success probabilities of $\mathcal{A}$ in each of the games G_0 – G_{12}, we get

$$\mathsf{Adv}^{\mathsf{ind-cca}}_{\Pi_h,\mathcal{A}}(t) \leq \mathsf{Adv}^{\mathsf{ind-cpa}}_{\Pi_s,\mathcal{B}}(t) + \delta + 4q_1\sqrt{\frac{q_3}{\sqrt{|\mathcal{M}_p|}} + \frac{q_d}{2^{|G|}} + \delta' + \mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi_p,\mathcal{D}}(t) + 2q_2\sqrt{\delta' + \mathsf{Adv}^{\mathsf{ow-cpa}}_{\Pi_p,\mathcal{E}}(t)}}$$

3.3 Threshold Variant

Assuming there are protocols $\Pi_{\mathcal{K}_p}$ and $\Pi_{\mathcal{D}_p}$ which implement the base public key encryption scheme in a threshold manner a threshold variant of our above constructions are therefore immediate. We simply apply the threshold decryption operation to c_1^*, keeping the result in a shared form. The parties then securely evaluate G (or G, H' and H'' in our second hybrid construction). Our distributed decryption operation for our Hybrid_1 construction Π_h would then consist of the following steps, with a similar methodology for Hybrid_2 (which would also require a secure evaluation of H' and H'')

1. Absorb c_2 (resp. c_1 and c_2) into G in the clear.
2. Apply $\Pi_{\mathcal{D}_p}$ to obtain a distributed decryption operation, keeping the result k in shared form.
3. Securely absorb these shares of k into the sponge G.
4. Securely evaluate the squeezing of G to obtain c_3' in the clear.
5. Reject the ciphertext if $c_3 \neq c_3'$.
6. Open k to all players.

7. Compute $\mathsf{k} = H(k)$ in the clear
8. Compute $m = \mathcal{D}_s(\mathsf{k}, c_2)$ in the clear and output it.

We notice that if we use a sponge-like function for G, such as Rescue [AAB+19] or SHA-3, then in the clear we can insert the first arguments for G (c_1 and c_2) during a distributed decryption, as they are public. Thus we only need to execute a secure distributed version of G for the final absorbtion of k, and then the squeezing phase to obtain c_3.

4 Pre-Quantum Instantiation

We provide a pre-quantum instantiation of our modified KEM-DEM construction which enables one to produce threshold public key encryption for long messages. The key trick is to use an MPC-friendly hash function for the function G above; for example Rescue [AAB+19]. Here we focus on a discrete logarithm based construction. One methodology for discrete logarithms, as explained in the introduction, would be to adopt the TDH1 or TDH2 construction from [SG98,SG02] into a Tag-KEM system. This is simply done by interpreting the 'label' in TDH1 and TDH2 as the 'tag' (the hash of the DEM ciphertext) from the Tag-KEM framework.

Another variant, which uses our explicit hybrid construction, is to take the text-book ElGamal cryptosystem

$$(\ C_1 = [r]P,\ C_2 = M + [k]Q\ )$$

over an elliptic curve $E(\mathbb{F}_p)$ with base-point P of prime order q and for public key $Q = [d]P$. This is because text-book ElGamal is an OW-PCA encryption scheme under the Gap-Diffie-Hellman assumption, and thus we can apply our hybrid construction. Focusing on full threshold for simplicity, one can split the decryption key d into $d = d_1 + \ldots + d_n \pmod q$. A passively secure distributed decryption for Π_p is then for each party to output $M_i \leftarrow [d_i]C_1$ and then compute $M \leftarrow C_2 - M_1 - \ldots - M_n$. This can be made actively secure using standard zero-knowledge proofs, if one wanted a stand alone actively secure variant of Π_p.

If we take an instantiation of Rescue modulo p for the function G then a distributed decryption operation for Π_h would then consist of the following steps, assuming an MPC system which works modulo p. Note, we need to pass c_1 into the sponge G as a Π_p is a randomized IND-PCA encryption scheme and not deterministic in this case.

1. Absorb c_1 and c_2 into G in the clear using some mapping of bit strings into elements modulo p for the absorbtion of c_2.
2. Locally compute $M_i = (X_i, Y_i) \leftarrow [d_i]C_1$ in a passive manner.
3. Enter $M_i = (X_i, Y_i) \in \mathbb{F}_p^2$ into the MPC engine as $\langle M_i \rangle$.
4. Compute securely $\langle M \rangle \leftarrow C_2 - \langle M_1 \rangle - \ldots - \langle M_n \rangle$.
5. Securely absorb $\langle M \rangle$ into the sponge G.
6. Securely evaluate the squeezing of G to obtain g in the clear.
7. Reject the ciphertext if $c_3 \neq g$.
8. Open $k = M$ to all players.
9. Compute $\mathsf{k} \leftarrow H(k)$ in the clear
10. Compute $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$ in the clear and output it.

Note here we can get an actively secure distributed decryption operation without needing zero-knowledge proofs to imply the correctness of the values M_i. To see this, notice that the values M_i

are only ever input into the MPC engine, they are never transmitted to the decrypting parties. Upon entry into the MPC system, assuming it is actively secure, the simulator can extract the values M_i enterred by the adversary and hence perfectly simulate the honest players interaction to the adversary. We leave the details to the reader. This is unlikely to be more efficient than using TDH1 or TDH2, however, the general principle here will serve as a motivation for our post-quantum construction later.

The Gladius–Hispaniensis Deterministic Encryption Scheme Π_p.

Key Generation: $\mathcal{K}_p$.
1. $R_1, R_2 \leftarrow \mathcal{D}_\sigma^{n\times n}$, i.e. two $n \times n$ matrices with coefficients samples from $\mathcal{D}_\sigma$.
2. $A_1 \leftarrow \mathbb{Z}_q^{n\times n}$
3. $A_2 \leftarrow A_1 \cdot R_1 + R_2 + G$, where G is the gadget matrix $\ell \cdot I_n$.
4. $\mathsf{pk} \leftarrow (A_1, A_2)$.
5. $\mathsf{sk} \leftarrow (\mathsf{pk}, R_1)$.
6. Return $(\mathsf{pk}, \mathsf{sk})$).

Encryption: $\mathcal{E}_p(\mathsf{pk}, \mathbf{m})$.
1. $\mathbf{c}_1 \leftarrow \lfloor \mathbf{m}^\mathsf{T} \cdot A_1 \rceil_p$.
2. $\mathbf{c}_2 \leftarrow \lfloor \mathbf{m}^\mathsf{T} \cdot A_2 \rceil_p$.
3. Return $(\mathbf{c}_1, \mathbf{c}_2)$.

Decryption: $\mathcal{D}_p(\mathsf{sk}, (\mathbf{c}_1, \mathbf{c}_2))$.
1. $\mathbf{w}^\mathsf{T} \leftarrow \mathbf{c}_2 - \mathbf{c}_1 \cdot R_1 \pmod q$
2. $\mathbf{e}^\mathsf{T} \leftarrow \mathbf{w}^\mathsf{T} \pmod p$.
3. $\mathbf{v}^\mathsf{T} \leftarrow \mathbf{e}^\mathsf{T} \pmod \mu$.
4. $\mathbf{m}^\mathsf{T} \leftarrow (\mathbf{e}^\mathsf{T} - \mathbf{v}^\mathsf{T})/\mu$.
5. $(\mathbf{c}_1', \mathbf{c}_2') \leftarrow \mathcal{E}_p(\mathsf{pk}, \mathbf{m})$.
6. If $\mathbf{c}_1 \neq \mathbf{c}_1'$ or $\mathbf{c}_2 \neq \mathbf{c}_2'$ return $\perp$.
7. Return $\mathbf{m}^\mathsf{T}$.

Figure 8. The Gladius–Hispaniensis Deterministic Encryption Scheme Π_p.

5 Gladius–Hispaniensis: Plain LWR Based Encryption

According to Wikipedia the *Gladius–Hispaniensis* was the earliest and heaviest of the different types of Gladii that we know about; it is thus fitting we reserve this name for our encryption scheme based on standard LWR. The scheme is defined in Figure 8 and is parametrized by values $t, p, q, n, \ell, \sigma, \epsilon$. We define the message space $\mathcal{M}$ to be the set $\mathbb{Z}_t^n$. From these parameters we define $\mu \in \mathbb{Z}$ and $\psi \in (-1/2, 1/2]$ via

$$\frac{p \cdot \ell}{q} = \left\lfloor \frac{p \cdot \ell}{q} \right\rceil + \psi = \mu + \psi. \tag{2}$$

Note when μ and p are powers of two, say $\mu = 2^\nu$ and $p = 2^\pi$, and $t = 2$ then lines 3 and 4 of the decryption procedure in Figure 8 becomes $m_i \leftarrow w_i^{(\nu)} \oplus w_i^{(\nu+1)}$, where $\mathbf{m} = (m_i)$ and $\mathbf{w} = (w_i)$ and $w_i^{(j)}$ is the j-th bit of w_i. This is again a useful simplification in our distribtued decryption procedure, thus we will assume that μ and p are powers of two.

Correctness: To provide conditions which need to be satisfied in order to ensure correct decryption, we compute the probability of a decryption failure as we vary over the entire message space, i.e. our probability space is the set of all $\mathbf{m}$ modelled as vectors chosen uniformly at random.

We express the ciphertext components as equations over the rationals as, where $\mathbf{e}_i \in \mathbb{R}^m$ with $\|\mathbf{e}_i\|_\infty \leq 1/2$ and $\mathbf{v}_i \in \{0,1\}^m$,

$$\begin{aligned}
\mathbf{c}_1 &= \frac{p}{q} \cdot \Big(\mathbf{m}^\mathsf{T} \cdot A_1 \pmod q \Big) \;+\; \mathbf{e}_1^\mathsf{T} \;+\; p \cdot \mathbf{v}_1^\mathsf{T}, \\
\mathbf{c}_2 &= \frac{p}{q} \cdot \Big(\mathbf{m}^\mathsf{T} \cdot A_2 \pmod q \Big) \;+\; \mathbf{e}_2^\mathsf{T} \;+\; p \cdot \mathbf{v}_2^\mathsf{T} \\
&= \frac{p}{q} \cdot \Big(\Big(\mathbf{m}^\mathsf{T} \cdot A_1 \pmod q \Big) \cdot R_1 + \mathbf{m}^\mathsf{T} \cdot (R_2 + G) + q \cdot \mathbf{r}^\mathsf{T} \Big) \;+\; \mathbf{e}_2^\mathsf{T} \;+\; p \cdot \mathbf{v}_2^\mathsf{T}.
\end{aligned}$$

Note, that if we consider the distribution of $\mathbf{e}_i$ then it acts as an independent random variable with entries chosen with mean zero and variance bounded by $1/12$. The entries of $\mathbf{v}_i$ are zero with probability $1 - 1/p$, so we estimate the variance of the entries of $\mathbf{v}_i$ as $(1-1/p)/p$. We also see that the entries of $\mathbf{m}$ have mean zero and variance $((t+1)^2 - 1)/12$. The vector $\mathbf{r}$ will have variance around $n^2 \cdot \sigma^2 \cdot ((t+1)^2 - 1)/144$. Note, when q is a power-of-two, since $p|q$, the vector $\mathbf{r}$ plays no role in decryption and thus in this case we have a simpler analysis to perform. For this reason, we set $c_q = 1$ when q is *not* a power-of-two.

These estimates of variances imply, given our earlier discussion on probabilities, that we have with probability approximately $1 - 4 \cdot n \cdot 2^{-\epsilon}$ that all the following four inequalities are satisfied

$$\begin{aligned}
\|\mathbf{m}^\mathsf{T} \cdot R_2\|_\infty &\leq \mathfrak{c} \cdot \sigma \cdot \sqrt{n \cdot ((t+1)^2 - 1)/12}, \\
\|\mathbf{e}_1^\mathsf{T} \cdot R_1\|_\infty &\leq \mathfrak{c} \cdot \sigma \cdot \sqrt{n/12}, \\
\|\mathbf{v}_1^\mathsf{T} \cdot R_1\|_\infty &\leq \mathfrak{c} \cdot \sigma \cdot \sqrt{\frac{(1-1/p) \cdot n}{p}} = \mathfrak{c} \cdot \frac{\sigma}{p} \cdot \sqrt{(p-1) \cdot n}, \\
\|\mathbf{r}^\mathsf{T}\|_\infty &\leq \mathfrak{c} \cdot n \cdot \sigma \cdot \sqrt{(t+1)^2 - 1}/12
\end{aligned}$$

where we take the probability space over all possible public keys and (uniformly random) messages.

To simplify the analysis we will write

$$\begin{aligned}
\mathbf{v}^\mathsf{T} &= \frac{p}{q} \cdot \mathbf{m}^\mathsf{T} \cdot R_2 \;+\; \psi \cdot \mathbf{m}^\mathsf{T} \;+\; \mathbf{e}_2^\mathsf{T} \;-\; \mathbf{e}_1^\mathsf{T} \cdot R_1, \\
\mathbf{u}^\mathsf{T} &= \mathbf{v}_2^\mathsf{T} - \mathbf{v}_1^\mathsf{T} \cdot R_1 + c_q \cdot \mathbf{r}^\mathsf{T}.
\end{aligned}$$

We can then write the first part of the decryption procedure as

$$\begin{aligned}
\mathbf{w}^\mathsf{T} &= \mathbf{c}_2 - \mathbf{c}_1 \cdot R_1 \pmod q \\
&= \frac{p}{q} \cdot \Big(\Big(\mathbf{m}^\mathsf{T} \cdot A_1 \pmod q \Big) \cdot R_1 + \mathbf{m}^\mathsf{T} \cdot (R_2 + G) + c_q \cdot q \cdot \mathbf{r}^\mathsf{T} \Big) \\
&\qquad + \; \mathbf{e}_2^\mathsf{T} \;+\; p \cdot \mathbf{v}_2^\mathsf{T} \\
&\qquad - \frac{p}{q} \cdot \Big(\mathbf{m}^\mathsf{T} \cdot A_1 \pmod q \Big) \cdot R_1 \;-\; \mathbf{e}_1^\mathsf{T} \cdot R_1 \;-\; p \cdot \mathbf{v}_1^\mathsf{T} \cdot R_1 \pmod q \\
&= \frac{p}{q} \cdot \mathbf{m}^\mathsf{T} \cdot R_2 \;+\; \frac{p}{q} \cdot \mathbf{m}^\mathsf{T} \cdot G \;+\; \mathbf{e}_2^\mathsf{T} \;-\; \mathbf{e}_1^\mathsf{T} \cdot R_1 \\
&\qquad + \; p \cdot (\; \mathbf{v}_2^\mathsf{T} - \mathbf{v}_1^\mathsf{T} \cdot R_1 + c_q \cdot \mathbf{r}^\mathsf{T} \;) \pmod q \\
&= \frac{p}{q} \cdot \mathbf{m}^\mathsf{T} \cdot R_2 \;+\; \mu \cdot \mathbf{m}^\mathsf{T} \;+\; \psi \cdot \mathbf{m}^\mathsf{T} \;+\; \mathbf{e}_2^\mathsf{T} \;-\; \mathbf{e}_1^\mathsf{T} \cdot R_1
\end{aligned}$$

$$
\begin{aligned}
&+\ p \cdot (\ \mathbf{v}_2^{\mathsf{T}} - \mathbf{v}_1^{\mathsf{T}} \cdot R_1 + c_q \cdot \mathbf{r}^{\mathsf{T}}\ ) \pmod q \\
&= \mathbf{v}^{\mathsf{T}} + \mu \cdot \mathbf{m}^{\mathsf{T}} + p \cdot \mathbf{u}^{\mathsf{T}} \pmod q
\end{aligned}
$$

In this equation $\mathbf{v}$ is the only part of the equation which consists of non-integral values, but since the whole equation is integral we know this part evaluates to an integer vector. By above, with probability $1 - 4 \cdot n \cdot 2^{-\epsilon}$, we know that we have

$$
\begin{aligned}
\|\mathbf{v}^{\mathsf{T}}\|_\infty \leq B_V &= \frac{p}{q} \cdot \mathfrak{c} \cdot \sqrt{n \cdot ((t+1)^2 - 1)/12} \cdot \sigma \ +\ \frac{t}{4} \ +\ \frac{1}{2} \ +\ \mathfrak{c} \cdot \sqrt{n/12} \cdot \sigma \\
&= \frac{t+2}{4} \ +\ \mathfrak{c} \cdot \sigma \cdot \sqrt{n} \cdot \left(\frac{p}{q} \cdot \sqrt{((t+1)^2 - 1)/12} \ +\ \sqrt{1/12} \right). \\
\|\mathbf{u}^{\mathsf{T}}\|_\infty \leq B_U &= 1 + \mathfrak{c} \cdot \sqrt{(p-1) \cdot n} \cdot \frac{\sigma}{p} \ +\ c_q \cdot \mathfrak{c} \cdot n \cdot \sigma \cdot \sqrt{(t+1)^2 - 1}/12 \\
&= 1 + \mathfrak{c} \cdot \sigma \cdot \left(\frac{\sqrt{(p-1) \cdot n}}{p} \ +\ c_q \cdot n \cdot \sqrt{(t+1)^2 - 1}/12 \right)
\end{aligned}
$$

Our first requirement is that $\|\mathbf{v}^{\mathsf{T}} + \mu \cdot \mathbf{m}^{\mathsf{T}} + p \cdot \mathbf{u}^{\mathsf{T}}\|_\infty$ is less than $q/2$, so there is no wrap-around modulo q when computing $\mathbf{w}^{\mathsf{T}}$ in this way. Remember we use the centred distribution always so, for example, we have $\|\mathbf{m}^{\mathsf{T}}\|_\infty \leq t/2$. Thus to ensure this happens we can, with probability $1 - 4 \cdot n \cdot 2^{-\epsilon}$, ensure that

$$
\|\ \mathbf{v}^{\mathsf{T}} + \mu \cdot \mathbf{m}^{\mathsf{T}} + p \cdot \mathbf{u}^{\mathsf{T}}\ \|_\infty \leq B_W = B_V + \frac{t}{2} \cdot \mu + p \cdot B_U \leq q/2. \tag{3}
$$

For our second requirement, we require that the third part of the decryption procedure also does not result in a wrap around. Thus we need the infinity norm of $\mathbf{v}^{\mathsf{T}}$ to be bounded by

$$
B_V < \mu/2. \tag{4}
$$

As our final requirement we need that the second part of the decryption procedure also does not result in a wrap around. Hence, we require

$$
\|\mathbf{e}^{\mathsf{T}}\|_\infty = \|\mathbf{v}^{\mathsf{T}} + \mu \cdot \mathbf{m}^{\mathsf{T}}\|_\infty \leq B_E = B_V + \mu \cdot \frac{t}{2} < p/2. \tag{5}
$$

Due to equation (3) we can write the decryption equation *over the integers*, and not modulo q, as

$$
\mathbf{w}^{\mathsf{T}} = \mathbf{v}^{\mathsf{T}} + \mu \cdot \mathbf{m}^{\mathsf{T}} + p \cdot \mathbf{u}^{\mathsf{T}}.
$$

Then, due to equation (5), we have that $\mathbf{w}^{\mathsf{T}} \pmod p$, when we take the centred reduction modulo p is exactly equal to $\mathbf{e}^{\mathsf{T}} = \mathbf{v}^{\mathsf{T}} + \mu \cdot \mathbf{m}^{\mathsf{T}}$ over the integers. Finally, due to equation (4), if we take $\mathbf{e}^{\mathsf{T}} \pmod \mu$, again using the centered reduction, we obtain exactly $\mathbf{v}^{\mathsf{T}}$ over the integers. Thus the decryption procedure exactly recovers $\mathbf{v}^{\mathsf{T}}$ and $\mathbf{e}^{\mathsf{T}}$, and hence $\mathbf{m}^{\mathsf{T}}$.

Summing up if equations (3), (4) and (5) hold then the probability of decryption error is approximately $4 \cdot n \cdot 2^{-\epsilon}$. Setting $\epsilon = 128$ will produce an error small enough for all practical purposes, for all values of n we will be considering. As remarked earlier, with $\epsilon = 128$ we need to select $\mathfrak{c} = 9.3$.

OW-CPA Security: OW-CPA security of the constructions rests on two computational assumptions. The first is that the public keys (A_1, A_2) are indistinguishable from uniformly random matrices. This will follow from the standard LWE assumption $\mathsf{LWE}_{q,n,\sigma}$, for which we can derive constraints on the parameters using the previously mentioned LWE-estimator. The second assumption is that the ciphertexts are also one-way; note we will not require indistinguishability in our application, which is the standard one-way LWR problem. OW-CPA security then follows immediately from these assumptions.

Parameters: Combining the correctness requirements, with the security requirement that the scheme is OW-CPA can be done in two different ways. Either we can use the theoretical reduction from one-way LWR to one-way LWE, given in Theorem 2.1, or we can estimate the parameters using the best-attack scenario. The former is better in theory, whilst the latter is better in practice. Indeed the latter is what is done for the estimation of parameters for the NIST Post-Quantum candidates based on LWR.

To select parameters we need to define the required search space to ensure both security and correctness. Recall we need to select parameters $t, p, q, n, \ell, \sigma, \epsilon$. As mentioned earlier we select $\epsilon = 128$ and so $\mathfrak{c} = 9.3$ in our earlier correctness equations. We select $\sigma = \sqrt{1/2}$ as this provides the simplest and cheapest parameters for deriving the secret key via the NewHope approximation to a discrete Gaussian. We select $t = 2$, so that we encrypt messages which are bit vectors, and select p and ℓ to be powers to two to enable more efficient modular reduction. As remarked earlier the value q is then (preferably) chosen so that μ in equation (2) is also a power-of-two. The value q is selected either to be a power-of-two, which gives performance advantages (since $c_q = 0$ in this case) when threshold variants are not of interest, or q is selected to be prime. We also look at two power values of n and general values of n.

Suppose n is fixed, then the search space is not big. When q is a power of 2, this is obvious. When q is a prime, we are only interested in, in fact, primes that are "close" to a power-of-two for efficiency reasons during distributed decryption explained later in Section 8, so the search space is also small and we can iterate over all the possibilities.

Then we use the bisection algorithm to iteratively find the smallest n that gives us enough security, assuming our correctness equations are satisfied. This works for a general n as well as a power-of-two value of n. To check security using Albrecht's tool, we use $\sigma = \sqrt{1/2}$ for the LWE security parameters that correspond to the public key. Due to Theorem 2.1 we use $B = \mathfrak{c} \cdot \sigma$ for the case when we measure LWR security via the security reduction. Finally, we use $\sigma' = \sqrt{\frac{(q/p)^2-1}{12}}$ when checking security for the LWR case in the 'best-attack' estimation using Albrecht's tool.

We use the the minimum between the LWE security estimate and the LWR 'best-attack' security estimate (like the NIST Post-Quantum candidates) as the overall security. Concretely, the primal attack (uSVP version) is used as the reduction cost model in Albrecht's tool. From the security estimates found in [ACD+18], this model appears to be the more pessimistic choice when comparing with the dual attack. Table 1 shows parameter choices for the two types of q, being a prime and being a power of 2, and two security targets, namely 128 and 256-bit security, and a choice of n a power-of-two, and one where n can be any value. Note, the theoretical hardness for the $q = 2^k$ case is very low due to the denominator in Theorem 2.1 being so large in this case, as the ratio p/q is much larger. This gives us the parameters in the following table.

δ_c-Collision Freeness: We note that whilst Gladius is not perfectly correct, it is collision free with very high probability. For a fixed public/private key pair $(\mathsf{pk}, \mathsf{sk})$ we can upper bound the

	n	t	q	p	ℓ	σ	μ	LWE Security	LWR Security Theoretical	LWR Security Best-Attack
prime q	971	2	$2^{21}-9$	2^9	2^{19}	$\sqrt{1/2}$	128	$2^{128.3}$	$2^{61.25}$	$2^{465.7}$
	1024	2	$2^{21}-9$	2^9	2^{19}	$\sqrt{1/2}$	128	$2^{135.7}$	$2^{64.78}$	$2^{492.7}$
	1982	2	$2^{23}-15$	2^{10}	2^{21}	$\sqrt{1/2}$	256	$2^{256.6}$	$2^{125.3}$	$2^{465.7}$
	2048	2	$2^{23}-15$	2^{10}	2^{21}	$\sqrt{1/2}$	256	$2^{266.0}$	$2^{129.9}$	$2^{975.5}$
$q=2^k$	710	2	2^{14}	2^{10}	2^{11}	$\sqrt{1/2}$	128	$2^{128.9}$	$2^{-550.1}$	$2^{187.6}$
	1024	2	2^{14}	2^{10}	2^{12}	$\sqrt{1/2}$	256	$2^{188.4}$	$2^{-792.1}$	$2^{274.8}$
	1437	2	2^{15}	2^{11}	2^{12}	$\sqrt{1/2}$	256	$2^{256.6}$	$2^{-1115.}$	$2^{366.1}$
	2048	2	2^{15}	2^{11}	2^{12}	$\sqrt{1/2}$	256	$2^{376.6}$	$2^{-1584.}$	$2^{535.3}$

Table 1. Gladius–Hispaniensis parameters (based on plain LWR)

probability of *any* collision existing as follows. The encryption scheme is a mapping from $\mathcal{M} = \mathbb{Z}_t^n$ to the set $\mathbb{Z}_p^{2n}$. Indeed under the decision-LWR assumption this map is indistinguishable from a random mapping. Thus to estimate δ_c we need to estimate the probability that this specific random mapping (defined by the public/private key pair) has any collisions at all.

We can then consider the δ_c-Collision Freeness as being given by the probability that one obtains a collision when sampling t^n values randomly from a set of size p^{2n}. The standard analysis of the birthday-paradox estimates this probability as

$$\delta_c = 1 - \exp\left(-t\cdot(t-1)/(2\cdot p^{2n})\right) \approx 1 - \exp\left(-(t/p)^{2n}/2\right).$$

For our specimen parameters this value is less than 2^{-15000}.

$\perp$-Aware *Hardness:* Recall $\perp$-Aware security is also related to the correctness of the scheme, it requires that it is hard for an adversary to write down a message/ciphertext pair (m, c) such that c is an encryption of m, but c decrypts to $\perp$. To understand $\perp$-Aware security of our scheme we focus on the *specific* parameter sets given above (again a similar analysis also applies in the case of the Module-LWR parameters given later).

If instead of averaging over all public keys and messages to obtain our initial four bounds, we average over all public keys and the worst case for messages, we obtain the following bounds,

$$\begin{aligned}
\|\mathbf{m}^\mathsf{T}\cdot R_2\|_\infty &\le \mathfrak{c}\cdot\sigma\cdot\sqrt{n}\cdot\frac{t}{2},\\
\|\mathbf{e}_1^\mathsf{T}\cdot R_1\|_\infty &\le \mathfrak{c}\cdot\sigma\cdot\sqrt{n}\cdot\frac{1}{2},\\
\|\mathbf{v}_1^\mathsf{T}\cdot R_1\|_\infty &\le \mathfrak{c}\cdot\sigma\cdot\sqrt{n},\\
\|\mathbf{r}^\mathsf{T}\|_\infty &\le \mathfrak{c}_2\cdot n\cdot\sigma\cdot\frac{t}{2\cdot\sqrt{12}}
\end{aligned}$$

with probability $1-4\cdot n\cdot 2^{-\epsilon}$, where $\mathfrak{c}$ and $\mathfrak{c}_2$ are such that $\mathsf{erfc}(\mathfrak{c}) \approx \mathsf{erfc}(\mathfrak{c}_2)^2 \approx 2^{-\epsilon}$. Passing these new bounds through the previous analysis we obtain

$$\begin{aligned}
\|\mathbf{v}^\mathsf{T}\|_\infty \le B_V &= \frac{p}{q}\cdot\mathfrak{c}\cdot\sigma\cdot\sqrt{n}\cdot\frac{t}{2} \;+\; \frac{t}{4} \;+\; \frac{1}{2} \;+\; \mathfrak{c}\cdot\sqrt{n}\cdot\sigma\cdot\frac{1}{2}\\
&= \frac{t+2}{4} \;+\; \mathfrak{c}\cdot\sigma\cdot\sqrt{n}\cdot\frac{p\cdot t+q}{2\cdot q}.
\end{aligned}$$

$$\|\mathbf{u}^{\mathsf{T}}\|_\infty \leq B_U = 1 + \mathfrak{c} \cdot \sigma \cdot \sqrt{n} \ + \ c_q \cdot \mathfrak{c}_2 \cdot n \cdot \sigma \cdot \frac{t}{2 \cdot \sqrt{12}}$$
$$= 1 + \sigma \cdot \left(\mathfrak{c} \cdot \sqrt{n} \ + \ c_q \cdot \mathfrak{c}_2 \cdot n \cdot \frac{t}{2 \cdot \sqrt{12}} \right).$$

The three final bounds being the same, namely

$$B_W = B_V + \frac{t}{2} \cdot \mu + p \cdot B_U \leq q/2,$$
$$B_V < \mu/2,$$
$$B_E = B_V + \mu \cdot \frac{t}{2} < p/2$$

If we substitute our various parameter sets into these equations we find, assuming $\epsilon = 128$, that in the case of prime q the only inequality which is not satisfied is $B_V < \mu/2$, and for q a power-of-two we can have $B_W < q/2$ not satisfied or $B_V < \mu/2$ not satisfied.

Further analysis reveals that (in all cases when q is prime) the bound of $B_V < \mu/2$ would be satisfied if we had

$$\|\mathbf{e}_1^{\mathsf{T}} \cdot R_1\|_\infty \leq \mathfrak{c}' \cdot \sigma \cdot \sqrt{n}/2$$

for $\mathfrak{c}' = 5.565$ at the 128-bit security level and $\mathfrak{c}' = 7.935$ at the 256-bit security level (including the module-LWR based parameters given later). Thus any pair (m, c) for which c decrypts to $\perp$, must violate this bound.

We have the following theorem, which establishes (using the above constants $\mathfrak{c}'$ and our conjectured hardness of the Large Vector Problem), that the Gladius scheme is $\perp$-Aware for the parameter sets of interest.

Theorem 5.1. *For the Gladius parameter sets when q is prime if there is an adversary $\mathcal{A}$ against $\perp$-Aware then there is an adversary $\mathcal{B}$ against LVP such that*

$$\mathsf{Adv}^{\perp-\mathsf{Aware}}_{\Pi_p,\mathcal{A}}(t) = \mathsf{Adv}^{\mathsf{LVP}}_{\mathcal{B}}(n, \mathfrak{c}', \sigma)$$

for the above values of $\mathfrak{c}'$.

Proof. The adversary $\mathcal{B}$ takes as input (A_1, A_2) which we pass on as the public key to adversary $\mathcal{A}$. Adversary $\mathcal{A}$ returns with a message/ciphertext (m, c) from which one can extract the vector $\mathbf{e}_1$. The only way the pair (m, c) can satisfy the constraints $c = \mathcal{E}_p(\mathsf{pk}, m)$ and $\perp = \mathcal{D}_p(\mathsf{sk}, c)$ is if this value of $\mathbf{e}_1$ satisfies

$$\|\mathbf{e}_1^{\mathsf{T}} \cdot R_1\|_\infty \leq \mathfrak{c}' \cdot \sigma \cdot \sqrt{n}/2.$$

Thus $\mathbf{e}_1$ will be a solution to the Large Vector Problem.

We end this section by noting the methodology here. We used a relatively loose noise analysis so as to obtain a sufficient search space for both correctness and LWE/LWR security. Then we re-examined the correctness equations in more detail, for these *specific* parameter sets, so as to establish the required hardness of the $\perp$-Aware problem, and an estimate for the δ_c value.

The above describes solely the KEM-like component Π_p of our hybrid construction from Section 3. The DEM-like component Π_s can be any (one-time) IND-CPA cipher; for example a one-time pad or AES in CTR-mode. The remaining item to define is the associated hash function G (and in

the case of using Hybrid_2 the hash functions H' and H''). Recall G takes the ciphertext c_2 output from the DEM, and the key k which the KEM encapsulates, and produces the hash result $G(c_2, k)$. Here we focus solely on the case of prime q variants of Gladius.

In our construction, to aid distributed decryption, we construct G as in Figure 9, assuming we take the message modulus $t = 2$ in our above construction. Minor tweaks are needed in the case when $t \neq 2$. The construction makes use of Rescue with rate r satisfying $r \geq 2 \cdot \kappa / \lfloor \log_2 q \rfloor$, as well as SHA-3. The combined hash function can clearly be treated as a random oracle if one assumes SHA-3 and Rescue are themselves random oracles. In the final distributed decryption variant only lines 5 and 6 need to be performed in a secure manner (which are based on Rescue, which is an MPC-friendly hash function). Thus irrespective of how long the initial message is which is being encrypted, the number of applications of Rescue which need to be performed securely is given by $\lceil w/r \rceil + 1$. If we take parameters $\kappa = 128$, $n = 1024$ and $q = 2^{21} - 9$ then we have $r = 13$, $w = 52$ and the number of secure rounds of Rescue is five in order to absorb the key k and produce the output $G(c_2, k)$. For the case of Hybrid_2 we select H' and H'' based on Rescue as well.

The Hash Function $G(c_2, k)$.

On input of c_1, c_2 and $k \in \{0,1\}^n$.

1. Apply the SHA-3 hash function to c_2 to obtain a $2 \cdot \kappa$-bit string $\mathbf{s}$.
2. Parse $\mathbf{s}$ into r bit-strings $(s_1, \ldots, s_r)$ each of length $\lfloor \log_2 q \rfloor$. This is possible due to the choice of r.
3. Treat each s_i as an element of $\mathbb{F}_q$ and absorb the set $(s_1, \ldots, s_r)$ into a fresh Rescue state. This requires one application of the Rescue absorption phase. Note, this can all be done in the clear during threshold decryption as c_1 and c_2 are public.
4. Take the bit string $k \in \{0,1\}^n$ and parse again into bit-strings of length $\lfloor \log_2 q \rfloor$. This will produce $w = \lceil n / \lfloor \log_2 q \rfloor \rceil$ bit-strings $k_1, \ldots, k_w$, each of which we think of as elements in $\mathbb{F}_q$.
5. The w finite field elements $k_1, \ldots, k_w$ are absorbed into the Rescue state, this will require $\lceil w/r \rceil$ executions of the Rescue function. Since during distributed decryption k is not known at this stage, this absorption needs to be carried out securely.
6. Finally the output is obtained by squeezing our r output field elements from Rescue using a single application of the Rescue function.

Figure 9. The Hash Function $G(c_2, k)$.

For q a power-of-two a different methodology will be required. We know of no MPC-friendly hash function defined over rings of the form $\mathbb{Z}_{2^k}$. Thus for the case of power-of-two values of q it would seem one would need to use a standard sponge-based hash function (such as SHA-3), which would not be as amenable to threshold implementation via a generic MPC methodology.

6 Gladius–Pompeii: Module LWR Based Encryption

According to Wikipedia the *Gladius–Pompeii* was the most popular and smallest of the different types of Gladii; it is thus appropriate we reserve this name for our encryption scheme based on module-LWR; as module-LWR provides a smaller construction than standard LWR. The construction is very much the same as for the standard LWR variant, and it is given in Figure 10. It is parametrized by values $t, p, q, n, d, \ell, \sigma, \epsilon$. From which we define μ as before by equation (2). Again for simplicity we select μ and p to be pwoers of two. We define the message space $\mathcal{M}$ to be the set $\mathcal{R}_t^d$.

The Gladius–Pompeii Deterministic Encryption Scheme Π_p.

Key Generation: Mod-$\mathcal{K}_p$.
1. $R_1, R_2 \leftarrow \mathcal{D}_\sigma(\mathcal{R}_q)^{d\times d}$, i.e. two $n \times n$ matrices with coefficients samples from $\mathcal{D}_\sigma$.
2. $A_1 \leftarrow \mathcal{R}_q^{d\times d}$
3. $A_2 \leftarrow A_1 \cdot R_1 + R_2 + G$, where G is the gadget matrix $\ell \cdot I_d$.
4. $\mathsf{pk} \leftarrow (A_1, A_2)$.
5. $\mathsf{sk} \leftarrow (\mathsf{pk}, R_1)$.
6. Return $(\mathsf{pk}, \mathsf{sk})$).

Encryption: Mod-$\mathcal{E}_p(\mathsf{pk}, \mathbf{m})$.
1. $\mathbf{c}_1 \leftarrow \lfloor \mathbf{m}^\mathsf{T} \cdot A_1 \rceil_p$.
2. $\mathbf{c}_2 \leftarrow \lfloor \mathbf{m}^\mathsf{T} \cdot A_2 \rceil_p$.
3. Return $(\mathbf{c}_1, \mathbf{c}_2)$.

Decryption: Mod-$\mathcal{D}_p(\mathsf{sk}, (\mathbf{c}_1, \mathbf{c}_2))$.
1. $\mathbf{w}^\mathsf{T} \leftarrow \mathbf{c}_2 - \mathbf{c}_1 \cdot R_1 \pmod q$
2. $\mathbf{e}^\mathsf{T} \leftarrow \mathbf{w}^\mathsf{T} \pmod p$.
3. $\mathbf{v}^\mathsf{T} \leftarrow \mathbf{e}^\mathsf{T} \pmod \mu$.
4. $\mathbf{m}^\mathsf{T} \leftarrow (\mathbf{e}^\mathsf{T} - \mathbf{v}^\mathsf{T})/\mu$.
5. $(\mathbf{c}_1', \mathbf{c}_2') \leftarrow$ Mod-$\mathcal{E}_p(\mathsf{pk}, \mathbf{m})$.
6. If $\mathbf{c}_1 \neq \mathbf{c}_1'$ or $\mathbf{c}_2 \neq \mathbf{c}_2'$ return $\perp$.
7. Return $\mathbf{m}^\mathsf{T}$.

Figure 10. The Gladius–Pompeii Deterministic Encryption Scheme Π_p.

The noise analysis proceeds as before: We can express the ciphertext components over $\mathcal{R} \otimes \mathbb{Q}$ as

$$\begin{aligned}
\mathbf{c}_1 &= \frac{p}{q} \cdot \Big(\mathbf{m}^\mathsf{T} \cdot A_1 \pmod q \Big) \ + \ \mathbf{e}_1^\mathsf{T} \ + \ p \cdot \mathbf{v}_1^\mathsf{T}, \\
\mathbf{c}_2 &= \frac{p}{q} \cdot \Big(\mathbf{m}^\mathsf{T} \cdot A_2 \pmod q \Big) \ + \ \mathbf{e}_2^\mathsf{T} \ + \ p \cdot \mathbf{v}_2^\mathsf{T} \\
&= \frac{p}{q} \cdot \Big(\Big(\mathbf{m}^\mathsf{T} \cdot A_1 \pmod q \Big) \cdot R_1 + \mathbf{m}^\mathsf{T} \cdot (R_2 + G) + q \cdot \mathbf{r}^\mathsf{T} \Big) \ + \ \mathbf{e}_2^\mathsf{T} \ + \ p \cdot \mathbf{v}_2^\mathsf{T}.
\end{aligned}$$

where $\mathbf{e}_i \in (\mathcal{R} \otimes \mathbb{Q})^d \subset \mathbb{R}^{n\cdot d}$ and $\|\mathbf{e}_i\|_\infty \leq 1/2$, and $\mathbf{v}_i \in \{0,1\}^{n\cdot d}$. We then set, as before, $\mathbf{w}^\mathsf{T} = \mathbf{v}^\mathsf{T} + \mu \cdot \mathbf{m}^\mathsf{T} + p \cdot \mathbf{u}^\mathsf{T} \pmod q$, where

$$\begin{aligned}
\mathbf{v}^\mathsf{T} &= \frac{p}{q} \cdot \mathbf{m}^\mathsf{T} \cdot R_2 \ + \ \psi \cdot \mathbf{m}^\mathsf{T} \ + \ \mathbf{e}_2^\mathsf{T} \ - \ \mathbf{e}_1^\mathsf{T} \cdot R_1, \\
\mathbf{u}^\mathsf{T} &= \mathbf{v}_2^\mathsf{T} - \mathbf{v}_1^\mathsf{T} \cdot R_1 + c_q \cdot \cdot \mathbf{r}^\mathsf{T}.
\end{aligned}$$

The previous analysis can then be applied, but we need to replace the value n from before with $n \cdot d$ now. In particular with probability $1 - 4 \cdot n \cdot d \cdot 2^{-\epsilon}$ we have the following bounds

$$\begin{aligned}
\|\mathbf{v}^\mathsf{T}\|_\infty &\leq B_V = \frac{t+2}{4} \ + \ \mathfrak{c} \cdot \sigma \cdot \sqrt{n \cdot d} \cdot \left(\frac{p}{q} \cdot \sqrt{((t+1)^2 - 1)/12} \ + \ \sqrt{1/12} \right), \\
\|\mathbf{u}^\mathsf{T}\|_\infty &\leq B_U = 1 + \mathfrak{c} \cdot \sigma \cdot \left(\frac{\sqrt{(p-1) \cdot n \cdot d}}{p} \ + \ c_q \cdot n \cdot d \cdot \sqrt{(t+1)^2 - 1}/12 \right), \\
\|\mathbf{e}^\mathsf{T}\|_\infty &\leq B_E = B_V + \mu \cdot \frac{t}{2}, \\
\|\mathbf{w}^\mathsf{T}\|_\infty &\leq B_W = B_E + p \cdot B_U
\end{aligned}$$

As before we require $B_W < q/2$, $B_V < \mu/2$ and $B_E < p/2$.

For security we again examine only the best attack case, where following the approach in [ACD+18] we approximate the hardness of solving module LWR and LWE for modules of rank d over rings of dimension d, as the same as the difficulty of solving normal LWR and LWE in dimension $d \cdot n$. Thus we seem to have the same bounds as we had for Gladius–Hispaniensis. However, with module LWR/LWE we can be more nuanced with the selection of d and n. We require n to be a power-of-two for the above noise analysis to work. We also require μ to be a power-of-two, which implies that q should be close to a power-of-two. To enable efficient arithmetic in the case when q is a prime we also require $q \equiv 1 \pmod{2 \cdot n}$, which will enable the ring arithmetic in $\mathcal{R}_q$ to be performed using a FFT representation. The extra flexibility though is that we have another free variable of d in the module setting. This leads us to the parameter sets in Table 2.

	n	d	t	q	p	ℓ	σ	μ	LWE Security	LWR Security Theoretical	LWR Security Best-Attack
prime q	256	4	2	2101249	2^9	2^{19}	$\sqrt{1/2}$	128	$2^{135.7}$	$2^{63.10}$	$2^{492.7}$
	256	8	2	8380417	2^{10}	2^{21}	$\sqrt{1/2}$	256	$2^{266.0}$	$2^{128.3}$	$2^{975.5}$
$q = 2^k$	256	3	2	2^{14}	2^{10}	2^{11}	$\sqrt{1/2}$	128	$2^{139.8}$	$2^{-594.8}$	$2^{203.4}$
	256	6	2	2^{15}	2^{11}	2^{12}	$\sqrt{1/2}$	256	$2^{275.7}$	$2^{-1192.}$	$2^{393.0}$

Table 2. Gladius–Pompeii parameters (based on module LWR)

We note that for the case of prime q the values are vary similar to those in the standard LWR based variant given earlier. When q is a power-of-two the parameters are much better than those in the equivalent standard LWR case. The parameter set $(q, n, d, p) = (2^{14}, 256, 3, 2^{10})$ looks very close to the Saber parameter set $(q, n, d, p) = (2^{13}, 256, 3, 2^{10})$. Thus run times for our entire construction for encryption and for (in the clear) decryption, will be comparable to those of Saber. However, the parameter set for Saber gives 256-bit security, whereas ours only gives 128-bit security due to the reliance on LWE for the key generation.

7 Gladius–Mainz: Potentially Easier Threshold Security in the QROM via a OTP

Our third Gladius variant provides (potentially) full threshold security in the QROM, at the expense of requiring a specific one-time IND-CPA DEM, which requires a distributed decryption procedure whose complexity is linear in the length of the message m. Thus it will not be truly efficient for long messages m. We dub this variant Gladius–Mainz, which is the name of a famous specific Gladius, sometimes called the 'Sword of Tiberius', to be found in the British Museum.

Recall our main hybrid construction Hybrid_1, in the case where Π_p, is a deterministic scheme can be considered as exactly the second Fujisaki–Okamoto transform when applied to the randomized encryption scheme

$$\mathcal{E}'_p(\mathsf{pk}, k; r) = (\mathcal{E}_p(\mathsf{pk}, k), r).$$

Note, here we think of Fujisaki–Okamoto as utilizing the randomization of a deterministic encryption scheme as opposed to the traditional way of thinking as it de-randomizing a randomized encryption scheme. Our encryption scheme of

$$k \leftarrow \mathcal{M}_p, \quad \mathsf{k} \leftarrow H(k), \quad c_1 \leftarrow \mathcal{E}_p(\mathsf{pk}, k), \quad c_2 \leftarrow \mathcal{E}_s(\mathsf{k}, m), \quad c_3 \leftarrow G(c_2, k).$$

for a deterministic OW-CPA secure Π_p *is* the Fujisaki–Okamoto transform; and thus by the result of Zhandry [Zha19] it is secure in the QROM when considered as a standard (non-threshold) encryption algorithm. But this result only holds when Π_p is perfectly secure. If we assume that this result can be extended to the case when Π_p has an exponential probability of decryption error then we can potentially obtain a simpler QROM secure construction.

We would like a QROM result, equivalent to the result of Theorem 3.1, for the threshold variant. As we do not want to evaluate the DEM in the secure domain, due to computational expense, we need to leak information. In our previous two constructions we leaked the key k from which the DEM key k was derived via application of H. Our first modification is to leak k instead of k. This we replace the leaking decryption oracle on the left with the one on the right

$\mathcal{D}_h(\mathsf{sk},(c_1,c_2,c_3))$:	$\mathcal{D}_h(\mathsf{sk},(c_1,c_2,c_3))$:
$k \leftarrow \mathcal{D}_p(\mathsf{sk},c_1)$	$k \leftarrow \mathcal{D}_p(\mathsf{sk},c_1)$
If $k = \perp$ then return $(\perp,\perp)$.	If $k = \perp$ then return $(\perp,\perp)$.
$t \leftarrow G(c_2,k)$	$t \leftarrow G(c_2,k)$
If $t \neq c_3$ then return $(\perp,\perp)$.	If $t \neq c_3$ then return $(\perp,\perp)$.
$\mathsf{k} \leftarrow H(k)$.	$\mathsf{k} \leftarrow H(k)$.
$m \leftarrow \mathcal{D}_s(\mathsf{k},c_2)$	$m \leftarrow \mathcal{D}_s(\mathsf{k},c_2)$
Return (k,m).	Return (k,m).

Our distributed decryption operation for Π_h would then consist of the following steps:

1. Absorb c_2 (resp. c_1 and c_2) into G in the clear.
2. Apply $\Pi_{\mathcal{D}_p}$ to obtain a distributed decryption operation, keeping the result k in shared form.
3. Securely absorb these shares of k into the sponge G.
4. Securely evaluate the squeezing of G to obtain g in the clear.
5. Reject the ciphertext if $c_3 \neq g$.
6. Securely evaluate $\mathsf{k} = H(k)$.
7. Open k to all players.
8. Compute $m = \mathcal{D}_s(\mathsf{k},c_2)$ in the clear and output it.

Thus we need to now evaluate the hash function H in the encrypted domain. This is not that much extra work for a choice of H similar to the choice of G, as long as the amount of data we need to squeeze from H is limited.

However, to show that the above construction is secure as a threshold scheme in the QROM we have to show that the information leaked by the threshold decryption is the same as is leaked by a normal decryption scheme. However, for general one-time IND-CPA schemes Π_s this is not going to be true. Since it implies that the key k can be derived from the plaintext-ciphertext pair (m,c_2).

We shall say the symmetric encryption scheme Π_s has a key-recovery algorithm (which we shall denote by $\mathcal{J}$) if $\mathcal{J}$ is a probabilistic polynomial time algorithm (measured in the length of k and/or m) such that

$$\Pr[\ \mathcal{J}(m,c) = k : \mathsf{k} \leftarrow \mathcal{K}_s,\ m \leftarrow \{0,1\}^*,\ r \leftarrow \mathcal{R},\ c \leftarrow \mathcal{E}_s(\mathsf{k},m;r)\ ] = 1.$$

So to obtain full QROM security we need to utilize a DEM with a key-recovery algorithm. But such a algorithm being available implies that an adversary can trivially break the IND-CPA game, unless the encryption scheme is the one-time-pad.

Thus the scheme will only be secure if a one-time-pad is used for Π_s, which implies that the hash function H needs to be securely evaluated to produce a key k as long as the message m. This

generic hybrid construction is QROM secure when we instantiate it with a scheme Π_p which is perfectly correct, by the result of Zhandry [Zha19].

However, when we instantiate Π_p with our LWR based deterministic encryption scheme we only obtain what we called QROM* security in the introduction; i.e. it is QROM secure assuming the proof of Zhandry can be extended to cope with non-perfectly correct schemes. We call Gladius–Pompeii when used in this manner Gladius–Mainz; it is clearly not as efficient when utilized in this specific threshold manner for long messages.

8 Distributed Decryption of Gladius

In this section we present how to perform distributed decryption of the hybrid cipher obtained from our generic construction composed with Gladius. For ease of implementation we select parameters for which q is prime, $p = 2^\pi$ and $\mu = 2^\nu$ are powers of two, and the message space modulus is $t = 2$. Although this section focuses on the simpler standard LWR variant (Gladius–Hispaniensis) and not on the Ring-LWR variants (Gladius–Pompeii, Gladius–Mainz and Gladius–Fulham), the procedure is virtually identical in all cases.

We use an MPC system defined for the q prime case for our experiments, as this is the only case for which we have both a full proof of security and a suitable MPC-friendly hash function (Rescue). Selecting q prime also means we can utilize an existing library such as SCALE-MAMBA [ACK$^+$20], for not only the underlying MPC system, but also many of the necessary sub-routines which our distributed decryption method requires. At the end of this section, we discuss the changes that would be required when q is a power-of-two.

We first present our distributed Key Generation protocol Π_{KeyGen}. Since the key generation method is based on Learning-with-Errors, with the error distribution coming from the NewHope distribution with $\sigma = 1/\sqrt{2}$, we can utilize the simple method described in [KLO$^+$19,RST$^+$19]. This is described in Figure 11.

Protocol for Distributed Key Generation Π_{KeyGen}.

1. For $i, j \in [1, \ldots, n]$
   - $\langle b \rangle, \langle b' \rangle, \langle c \rangle, \langle c' \rangle \leftarrow \mathsf{Bits}()$.
   - $\langle R_1^{(i,j)} \rangle \leftarrow \langle b \rangle - \langle b' \rangle$.
   - $\langle R_2^{(i,j)} \rangle \leftarrow \langle c \rangle - \langle c' \rangle$.
   - $A_1^{(i,j)} \leftarrow \mathbb{F}_q$.
2. $\langle A_2 \rangle \leftarrow A_1 \cdot \langle R_1 \rangle + \langle R_2 \rangle + G$.
3. $A_2 \leftarrow \mathbf{Output}(\langle A_2 \rangle)$.
4. $\mathsf{pk} \leftarrow (A_1, A_2)$.
5. $\mathsf{sk} \leftarrow (A_1, A_2, \langle R_1 \rangle)$.

Figure 11. Protocol for Distributed Key Generation Π_{KeyGen}.

The distributed decryption procedure itself is more complex. It makes use of the following protocols from other works, e.g. [DFK$^+$06,NO07]. In each of these protocols we *can* run the protocol with clear entries. For example $\mathsf{BitDecomp}(a)$ will form the bit decomposition of an integer a, but here we also need to specify how many bits we require. Since a may not necessarily be reduced in the range $(-q/2, \ldots, q/2)$. Thus we would write $\mathsf{BitDecomp}(a, t)$ to obtain t bits.

- $\langle \mathbf{a} \rangle \leftarrow \mathsf{BitDecomp}(\langle a \rangle)$: Given a secret shared value $\langle a \rangle$ with $a \in \mathbb{F}_q$ this procedure produces a vector of shared bits $\langle \mathbf{a} \rangle = (\langle a_0 \rangle, \ldots, \langle a_{\lfloor \log_2 q \rfloor} \rangle)$ such that $a = \sum_i a_i \cdot 2^i$. Note this means a is in the non-centred interval $[0, \ldots, q)$. The method we use is from [NO07], which is itself built upon the work in [DFK+06].
- $\langle \mathbf{c} \rangle \leftarrow \mathsf{BitAdd}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle)$: Given shared bits $\langle \mathbf{a} \rangle$ and $\langle \mathbf{b} \rangle$ this executes a binary adder to produce the vector of shared bits $\langle \mathbf{c} \rangle$ such that $\sum_i c_i \cdot 2^i = \sum_i (a_i + b_i) \cdot 2^i$. This algorithm is also presented in [DFK+06]. Note this returns one bit more than the maximum of the lengths of $\langle \mathbf{a} \rangle$ and $\langle \mathbf{b} \rangle$.
- $\langle \mathbf{c} \rangle \leftarrow \mathsf{BitNeg}(\langle \mathbf{a} \rangle)$: This performs the two-complement negative of the bit vector $\langle \mathbf{a} \rangle$. It flips the bits of $\langle \mathbf{a} \rangle$ to produce $\langle \overline{\mathbf{a}} \rangle$, and then executes the function $\mathsf{BitAdd}(\langle \overline{\mathbf{a}} \rangle, \mathbf{1})$, where $\mathbf{1} = \mathsf{BitDecomp}(1, |\overline{\mathbf{a}}|)$ is the bit-vector of the correct length representing the integer one.
- $\langle c \rangle \leftarrow \mathsf{BitLT}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle)$: This computes the single bit output $\langle c \rangle$ of the comparison $\sum_i a_i \cdot 2^i < \sum_i b_i \cdot 2^i$. Again we use the method from [DFK+06].

When running $\mathsf{BitDecomp}(\langle a \rangle)$ on a secret shared value the run time is not deterministic, it needs to loop to produce a shared value which is uniformly distributed modulo q. It does this by rejection sampling; where the probability of rejecting a sample is given by

$$\frac{2^{\lceil \log_2 q \rceil} - q}{2^{\lceil \log_2 q \rceil}}.$$

This is another reason to select q to be close to a power-of-two, as well as to ensure μ is a power of two.

In Figure 13 we divide our distributed decryption procedure into four phases: KEM Decapsulate, KEM Validity Check, the Hash-Check (for the checking of the DEM component) and finally the Message Extraction. As we select μ and p to be powers of two the first stage is relatively straightforward given we can implement $\mathsf{BitDecomp}(\langle a \rangle)$. There is a minor complication due to the need to map the bit-decomposition into the centred interval but this is easily dealt with using the sub-routine in Figure 12. The third stage complexity depends on the choice of the underlying hash function G; our choice of G from Section 5 using SHA-3 and Rescue combined was to ensure this step is as efficient as possible. Due to our hybrid design the final step can be performed in the clear; which is not possible for other hybrid schemes.

Thus, the main complexity of the decryption procedure is the second stage, namely the KEM Validity Check, as for this we need to re-encrypt the message and check the result is equal to the KEM ciphertext component. We need to verify equations of the following form

$$c = \lfloor \langle x \rangle \rceil_p = \left\lfloor \frac{p}{q} \cdot \langle x \rangle \right\rceil \pmod{p}$$

where c is publicly given, but the value $\langle x \rangle$ cannot be opened to the parties. We write the equation, over the integers, as

$$c = \frac{p}{q} \cdot \langle x \rangle \; + \; \epsilon \; + \; p \cdot v$$

where $\epsilon \in (-1/2, 1/2]$, $v \in \{0, 1\}$ and we think of the shared value $\langle x \rangle$ being in the centred representation modulo q. The value v is equal to one only if the reduction modulo p in the LWR equation needs to move the rounded value $-p/2$ to $p/2$. This happens when

$$x \leq \frac{q}{p} \left(\frac{1}{2} - \frac{p}{2} \right) = \frac{q \cdot (1 - p)}{2 \cdot p}.$$

This means we simply need to compute the bit representation $\langle \mathbf{s} \rangle$ of the value $|q \cdot c - p \cdot \langle x \rangle - p \cdot q \cdot \langle v \rangle|$ over the integers and then check the result is less than $q/2$. The last check can be performed using the $\mathsf{BitLT}(\langle \mathbf{s} \rangle, q/2)$ algorithm mentioned above.

But to compute the bit representation of $\langle \mathbf{s} \rangle$ we need the bit representation of the modulo q centred value $\langle x \rangle$. However, the BitDecomp routine only produces the bit-decomposition in the non-centred interval of a value modulo q. We could use the method from the first stage and apply the Centre sub-routine. However, this is inefficient as on its own it requires two calls to BitAdd (one explicitly to BitAdd and one implicitly in the call to BitNeg). The procedure BitAdd is our most expensive subroutine so we want to minimize the number of calls to this.

Thus instead we proceed as follows: If we think of the value $\langle x \rangle$ as the reduction in the centred interval, and $\langle u \rangle$ as the value in the non-centred interval then we have $x = u - b \cdot q$, where b is the bit given by $b = 1 - (u \leq q/2)$. We write $\langle \mathbf{u} \rangle$ for the corresponding shared bit decomposition of u. We can then re-write the equation for determining v above in terms of u, as opposed to x, as $v = b \cdot \left(u \leq \frac{q \cdot (p+1)}{(2 \cdot p)}\right)$. We note that $v = 0$ when $b = 0$, which is important in what follows.

We then rewrite the equation for $\langle \mathbf{s} \rangle$ as

$$\left| p \cdot \langle \mathbf{u} \rangle - p \cdot q \cdot (\langle b \rangle - \langle v \rangle) - c_i \cdot q \right|$$

The bit representation of $p \cdot \langle \mathbf{u} \rangle$ can be determined by simply shifting bits, as p is a power-of-two. The bit representation of $-p \cdot q \cdot (\langle b \rangle - \langle v \rangle)$ can be determined by bit-wise multiplications as $b - v \in \{0, 1\}$ by construction. From these observations we can produce a method for Stage 2 which requires three calls to BitAdd, as opposed to the naive method which would go through Centre which would require four calls to BitAdd.

Subroutine $\mathsf{Centre}(\langle x \rangle)$

1. $\langle \mathbf{b} \rangle \leftarrow \mathsf{BitDecomp}(\langle x \rangle)$; recall the bit-decomposition produces a value in the non-centred interval.
2. $\langle \mathbf{b}' \rangle \leftarrow \mathsf{BitAdd}(\langle \overline{\mathbf{b}} \rangle, q + 1)$; i.e. $b' = q - u_i$ over the integers.
3. $\langle \mathbf{b}'' \rangle \leftarrow \mathsf{BitNeg}(\langle \mathbf{b}' \rangle)$; i.e. $b'' = -b' \pmod{2^{\lceil \log_2 q \rceil} - \pi}$ if we compute to $\lceil \log_2 q \rceil - \pi$ bits.
4. $\langle f \rangle \leftarrow \mathsf{BitLT}(\langle \mathbf{b} \rangle, q/2)$; i.e. is $b < q/2$?
5. $\langle \mathbf{a} \rangle \leftarrow \langle f \rangle \cdot \langle \mathbf{b} \rangle \ + \ (1 - \langle f \rangle) \cdot \langle \mathbf{b}'' \rangle$. This is again done bitwise. This results in $\mathbf{a}$ being the bit representation of the centred value of u_i modulo q represented in $\lceil \log_2 q \rceil$ bits.
6. Return $\langle \mathbf{a} \rangle$.

Figure 12. Subroutine $\mathsf{Centre}(\langle x \rangle)$

Security Discussion and Implementation: As remarked previously the security of our implementation follows from the security of the underlying MPC protocol. By using SCALE-MAMBA [ACK+20] we can obtain active security, and the above sub-procedures are all provided as built in functions. In addition, the large local only operations in KEM Decapsulation (line 1) and KEM Validity Check (line 1) can be carried out efficiently in C++ using the SCALE `LOCAL_FUNCTION` operation. This enables one to perform complex local only operations, i.e. complex linear functions, natively in C++ as opposed to needing them to be implemented with the MPC system (which adds a lot of overhead).

Protocol for Distributed Decryption Π_{Dec}.

Input: A ciphertext $c_1 = (\mathbf{c}_1, \mathbf{c}_2), c_2, c_3$, the public key (A_1, A_2) and the secret key in shared form $\langle R_1 \rangle$.

KEM Decapsulation:

1. $\langle \mathbf{x} \rangle \leftarrow \mathbf{c}_2 - \mathbf{c}_1 \cdot \langle R_1 \rangle$.
2. For $i \in [1, \ldots, n]$
   - $\langle \mathbf{w} \rangle \leftarrow \mathsf{Centre}(\langle x_i \rangle)$.
   - $\langle k_i \rangle \leftarrow \langle w_i^{(\nu)} \rangle \oplus \langle w_i^{(\nu+1)} \rangle = \langle w_i^{(\nu)} \rangle \ + \ \langle w_i^{(\nu+1)} \rangle \ - \ 2 \cdot \langle w_i^{(\nu)} \rangle \cdot \langle w_i^{(\nu+1)} \rangle$.

KEM Validity Check:

1. $\langle \mathbf{y} \rangle \leftarrow \langle \mathbf{k} \rangle \cdot (A_1 \| A_2)$.
2. $\langle z \rangle \leftarrow 1$.
3. For $i \in [1, \ldots, 2 \cdot n]$
   - $\langle \mathbf{u} \rangle \leftarrow \mathsf{BitDecomp}(\langle y_i \rangle)$.
   - $\langle b \rangle \leftarrow 1 - \mathsf{BitLT}(\mathbf{u}, q/2)$.
   - $\langle v \rangle \leftarrow \langle b \rangle \cdot \mathsf{BitLT}(\langle \mathbf{u} \rangle, q \cdot (p+1)/(2 \cdot p))$. This computes the adjustment bit for dealing with the wrap around modulo p. Note, this can only apply when $a < 0$.
   - $\langle \mathbf{u}' \rangle \leftarrow \langle \mathbf{u} \rangle \ll \pi$; i.e. shift left by π bits, where $p = 2^\pi$. Hence $u' = p \cdot u$ over the integers, represented in $\lceil \log_2 q \rceil + \pi$ bits.
   - $\langle \mathbf{w} \rangle \leftarrow \mathsf{BitAdd}(\langle \mathbf{u}' \rangle, 2^{\lceil \log_2 q \rceil + \pi} - c_i \cdot q)$. Here $c_i = \mathbf{c}_1^{(i)}$ if $i \leq n$ and $\mathbf{c}_2^{(i-n)}$ otherwise. This produces $w = p \cdot u - c_i \cdot q$ over the integers with $\lceil \log_2 q \rceil + \pi$ bits.
   - $\langle \mathbf{f} \rangle \leftarrow \mathsf{BitAdd}(\langle \mathbf{w} \rangle, (\langle b \rangle - \langle v \rangle) \cdot (-p \cdot q))$. This applies the adjustment when $b = 1$ and $v = 0$. We now have $f = p \cdot u_i - (b - v) \cdot p \cdot q - c_i \cdot q$ over the integers with $\lceil \log_2 q \rceil + \pi$ bits.
   - $\langle \mathbf{f}' \rangle \leftarrow \mathsf{BitNeg}(\langle \mathbf{f} \rangle)$, hence $f' = -f$ over the integers.
   - $\langle g \rangle \leftarrow \langle f_{\pi + \lceil \log_2 q \rceil - 1} \rangle$; i.e. the sign bit of f.
   - $\langle \mathbf{s} \rangle \leftarrow \langle g \rangle \cdot \langle \mathbf{f}' \rangle \ + \ (1 - \langle g \rangle) \cdot \langle \mathbf{f} \rangle$. Again a bitwise operation computing $s = |f|$ as an integer.
   - $\langle j \rangle \leftarrow \mathsf{BitLT}(\langle \mathbf{s} \rangle, q/2)$; is one if this coefficient is OK.
   - $\langle z \rangle \leftarrow \langle z \rangle \cdot \langle j \rangle$; is one if the ciphertext is OK up to this point.
4. $z \leftarrow \mathbf{Output}(\langle z \rangle)$
5. If $z \neq 1$ then return $\perp$.

Hash Check:

1. $\langle t \rangle \leftarrow G(c_2, \langle \mathbf{k} \rangle)$.
2. $t \leftarrow \mathbf{Output}(\langle t \rangle)$.
3. If $t \neq c_3$ then return $\perp$.

Message Extaction:

1. $k \leftarrow \mathbf{Output}(\langle \mathbf{k} \rangle)$.
2. $\mathsf{k} \leftarrow H(k)$.
3. $m \leftarrow \mathcal{D}_s(\mathsf{k}, c_2)$
4. If $m = \perp$ then return $\perp$.
5. Return m.

Figure 13. Protocol for Distributed Decryption Π_{Dec}.

We implemented our distributed decryption procedure in the case of Shamir sharing within SCALE-MAMBA. This is because the Shamir implementation module allows the MPC sub-system to be instantiated over any finite field $\mathbb{F}_q$. In using a full threshold access structure one would need (with SCALE-MAMBA as currently implemented) to select a prime q which is FHE friendly; so as to enable the SHE scheme at the basis of SPDZ [DPSZ12] to be instantiated. None of the q values in the various parameter sets for Gladius are FHE friendly; not even the Gladius-Pompeii variants which have $q-1$ divisible by a large power of two.

For three parties, tolerating a threshold of one dishonest party, we obtained a run time for the first three phases of 1.19, 3.62, and 0.18 seconds respectively; for our parameter set of $q = 2^{21} - 9$ and $n = 1024$ in the plain LWR setting. Making a total decapsulation time of 4.99 seconds. Whilst this might at first sight seem slower than the 4.20 seconds reported for LIMA in [KLO$^+$19] the results are incomparable. Recall, the method in [KLO$^+$19] to perform distributed decapsulation is insecure, as indeed would be any distributed decapsulation of any algorithm making use of the traditional KEM-DEM construction.

Modifications When q is a Power-of-Two: Recall, we do not fully prove security of Gladius for the case when q is a power-of-two, since we cannot establish our $\perp$-Aware property in this case. Nevertheless, one can discuss how distributed decryption would be modified when q is a power-of-two. In this case, one could utilize (in the full threshold case) the SPDZ2k protocol of [CDE$^+$18]. When in the non-full threshold case one could implement an MPC system using replicated secret sharing to work modulo q a power-of-two, much like the Sharemind system [BLW08]. Active security could be obtained for such a replicated secret sharing based protocol using the techniques in [SW19]. Both replicated sharing in this case, and the SPDZ2k protocol, are implemented in the library MP-SPDZ [Kel20].

One immediate problem is how to implement the algorithm BitDecomp. Recall in our method above for the case of prime q we used bit-decomposition techniques which are secure absolutely (i.e. without any statistical security gap between the value being decomposed and the modulus q), these are more expensive but enable one to work in the MPC system natively modulo q. The techniques from [DFK$^+$06,NO07] are not known (by the authors) to have been generalized, or implemented, for the case of a general MPC system with q a power-of-two (using for example Replicated secret sharing). Thus techniques based on [Cd10], which uses a statistical security parameter, need to be adopted.

For the SPDZ2k full-threshold implementation in MP-SPDZ a method for performing bit-decomposition is provided in [DEF$^+$19]. This is itself based on the work in [Cd10], but unlike the latter it does not require a statistical security gap between the value being represented and the underlying modulus q. This is because SPDZ2k already has the statistical security 'gap' built into its computations.

Another advantage of using $\mathbb{Z}_{2^k}$ arithmetic is that we can work with a value of k large enough to cope with our integer arithmetic in the decryption test; thus we select k with $2^k > q$. For our q of 2^{14} one can select $k = 24$. Using MP-SPDZ for three parties, with the SPDZ2k protocol in the case of full-threshold adversaries and the 'Brain/Rep3/PS' method for replicated secret sharing in the honest majority case, we implemented the equivalent of the inner loop of line 3 of the Validity Check. We did not implement the full algorithm as MP-SPDZ does not have the `LOCAL_FUNCTION` ability which SCALE-MAMBA does; thus making implementation of line 1 less efficient than it could be.

Line 3 of the KEM Validity check for our Gladius-Hispaniensis parameters of $q = 2^{14}$, $p = 2^{10}$, $\mu = 128$, $\ell = 2^{11}$ using MP-SPDZ takes 0.272 seconds in the full threshold case, and 0.099 seconds in the replicated secret sharing case. For Gladius-Pompeii the time is 0.204 and 0.074 respectively, due to the reduced dimensions in the Module-LWR case. To this time would need to be added the time needed to compute line 1, which can be done relatively fast if performed in C++ directly. Recall the Validity Check step took 3.62 using SCALE-MAMBA for the case when q is a prime. Thus using q a power of two seems to give a roughly thirty-five fold performance improvement in the distributed decryption.

However, the major (implementation) problem for the case of q a power-of-two, is that we selected the hash function Rescue as it is MPC-friendly when q is prime. There has been no research work on designing MPC-friendly hash functions over rings of the form $\mathbb{Z}_{2^k}$, to our knowledge. Thus selection of the underlying hash function G in our construction could be a little more problematic. The obvious solution would be to utilize a standard hash function like SHA-3, but then one would need to pass from the secret sharing based MPC over $\mathbb{Z}_q$ to a garbled-circuit based MPC (such as HSS [HSS17]) in order to evaluate the SHA-3 operation efficiently. Whilst this is possible it would involve a considerable extra cost in terms of protocol complexity and execution time.

Thus, there are a number of interesting research challenges for implementing both a standard Gladius, and a distributed decryption procedure for Gladius, in the case when q is a power-of-two.

Acknowledgments

We would like to thank Alexandra Boldyreva for clarifying some issues with the PRIV definition of security for deterministic encryption, Frederik Vercauteren for clarifying some issues in relation to Learning-with-Rounding, Andrej Bogdanov for clarifying issues related to the theoretical reductions between LWE and LWR, and Ward Beullens on comments on an earlier draft.

This work was supported in part by CyberSecurity Research Flanders with reference number VR20192203, by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. FA8750-19-C-0502, and by the FWO under an Odysseus project GOH9718N. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC, DARPA, the US Government or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

AAB+19. Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Report 2019/426, 2019. https://eprint.iacr.org/2019/426.

ACD+18. Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the LWE, NTRU schemes! 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 351–367, Amalfi, Italy, September 5–7, 2018. Springer, Heidelberg, Germany.

ACK+20. Abdelrahaman Aly, Kelong Cong, Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Oliver Scherer, Peter Scholl, Nigel P. Smart, Titouan Tanguy, and Tim Wood. SCALE and MAMBA v1.9: Documentation, 2020.

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pages 327–343, Austin, TX, USA, August 10–12, 2016. USENIX Association.

AGK08. Masayuki Abe, Rosario Gennaro, and Kaoru Kurosawa. Tag-KEM/DEM: A new framework for hybrid encryption. *Journal of Cryptology*, 21(1):97–130, January 2008.

AKPW13. Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties and applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 57–74, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

AT09. Seiko Arita and Koji Tsurudome. Construction of threshold public-key encryptions through tag-based encryptions. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09: 7th International Conference on Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 186–200, Paris-Rocquencourt, France, June 2–5, 2009. Springer, Heidelberg, Germany.

BBO07. Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 535–552, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

BCLv19. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

BGM+16. Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 209–224, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

BLW08. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008: 13th European Symposium on Research in Computer Security*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206, Málaga, Spain, October 6–8, 2008. Springer, Heidelberg, Germany.

BP18. Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. Cryptology ePrint Archive, Report 2018/526, 2018. `https://eprint.iacr.org/2018/526`.

BST19. Charlotte Bonte, Nigel P. Smart, and Titouan Tanguy. Thresholdizing HashEdDSA: MPC to the Rescue. Cryptology ePrint Archive, Report 2020/214, 2019. `https://eprint.iacr.org/2020/214`.

Cd10. Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10: 7th International Conference on Security in Communication Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199, Amalfi, Italy, September 13–15, 2010. Springer, Heidelberg, Germany.

CDE+18. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 769–798, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

CS98. Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25, Santa Barbara, CA, USA, August 23–27, 1998. Springer, Heidelberg, Germany.

CS03. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

CS19. Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: Threshold post-quantum signatures. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *Lecture Notes in Computer Science*, pages 128–153, Oxford, UK, December 16–18, 2019. Springer, Heidelberg, Germany.

CS20. Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography -*

11th International Conference, PQCrypto 2020, pages 169–186, Paris, France, April 15–17 2020. Springer, Heidelberg, Germany.

DDN91. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, New Orleans, LA, USA, May 6–8, 1991. ACM Press.

DEF+19. Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy*, pages 1102–1120, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

Den03. Alexander W. Dent. A designer's guide to KEMs. In Kenneth G. Paterson, editor, *9th IMA International Conference on Cryptography and Coding*, volume 2898 of *Lecture Notes in Computer Science*, pages 133–151, Cirencester, UK, December 16–18, 2003. Springer, Heidelberg, Germany.

DFK+06. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

DKRV18. Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18: 10th International Conference on Cryptology in Africa*, volume 10831 of *Lecture Notes in Computer Science*, pages 282–305, Marrakesh, Morocco, May 7–9, 2018. Springer, Heidelberg, Germany.

DKRV19. Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

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, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

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, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

FO13. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013.

GKK+19. Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Starkad and Poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019. `https://eprint.iacr.org/2019/458`.

GZB+19. Oscar Garcia-Morchon, Zhenfei Zhang, Sauvik Bhattacharya, Ronald Rietman, Ludo Tolhuizen, Jose-Luis Torre-Arce, Hayo Baan, Markku-Juhani O. Saarinen, Scott Fluhrer, Thijs Laarhoven, and Rachel Player. Round5. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

Ham19. Mike Hamburg. Three Bears. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

HHK17. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

HSS17. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 598–628, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

IAHS07. Takeru Ishihara, Hiroshi Aono, Sadayuki Hongo, and Junji Shikata. Construction of threshold (hybrid) encryption in the random oracle model: How to construct secure threshold tag-KEM from weakly secure threshold KEM. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *ACISP 07: 12th Australasian Conference on Information Security and Privacy*, volume 4586 of *Lecture Notes in Computer Science*, pages 259–273, Townsville, Australia, July 2–4, 2007. Springer, Heidelberg, Germany.

JZC+18. Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 96–125, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

JZM19. Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Tighter security proofs for generic key encapsulation mechanism in the quantum random oracle model. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 227–248, Chongqing, China, May 8–10 2019. Springer, Heidelberg, Germany.

Kel20. Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. `https://eprint.iacr.org/2020/521`.

KLO+19. Michael Kraitsberg, Yehuda Lindell, Valery Osheter, Nigel P. Smart, and Younes Talibi Alaoui. Adding distributed decryption and key generation to a ring-LWE based CCA encryption scheme. In Julian Jang-Jaccard and Fuchun Guo, editors, *ACISP 19: 24th Australasian Conference on Information Security and Privacy*, volume 11547 of *Lecture Notes in Computer Science*, pages 192–210, Christchurch, New Zealand, July 3–5, 2019. Springer, Heidelberg, Germany.

KOR+17. Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Eduardo Soria-Vazquez, and Srinivas Vivek. Faster secure multi-party computation of AES and DES using lookup tables. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 229–249, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.

LL94. Chae Hoon Lim and Pil Joong Lee. Another method for attaining security against adaptively chosen ciphertext attacks. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 420–434, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

LLJ+19. Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, Zhenfei Zhang, Zhe Liu, Hao Yang, Bao Li, and Kunpeng Wang. LAC. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

LY12. Benoît Libert and Moti Yung. Non-interactive CCA-secure threshold cryptosystems with adaptive security: New framework and constructions. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 75–93, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

NAB+19. Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

NO07. Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 343–360, Beijing, China, April 16–20, 2007. Springer, Heidelberg, Germany.

NY90. Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

OP01. Tatsuaki Okamoto and David Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 159–175, San Francisco, CA, USA, April 8–12, 2001. Springer, Heidelberg, Germany.

PAA+19. Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. NewHope. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

RST+19. Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for SPDZ. Cryptology ePrint Archive, Report 2019/1300, 2019. `https://eprint.iacr.org/2019/1300`.

SAB+19. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

SG98. Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 1–16, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

SG02. Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, March 2002.

SW19. Nigel P. Smart and Tim Wood. Error detection in monotone span programs with application to communication-efficient multi-party computation. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, volume 11405 of *Lecture Notes in Computer Science*, pages 210–229, San Francisco, CA, USA, March 4–8, 2019. Springer, Heidelberg, Germany.

TU16. Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 192–216, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

Unr14. Dominique Unruh. Revocable quantum timed-release encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 129–146, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

XXZ12. Xiang Xie, Rui Xue, and Rui Zhang. Deterministic public key encryption and identity-based encryption from lattices in the auxiliary-input setting. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 1–18, Amalfi, Italy, September 5–7, 2012. Springer, Heidelberg, Germany.

ZCH+19. Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, William Whyte, John M. Schanck, Andreas Hulsing, Joost Rijneveld, Peter Schwabe, and Oussama Danba. NTRUEncrypt. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

Zha12. Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 758–775, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

Zha19. Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 239–268, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.