Hybrid Post-Quantum Additively Homomorphic Encryption & Private Information Retrieval

Jelle Vos[0000−0002−3979−9740]

mail@jelle-vos.nl

Abstract. We introduce the first generic constructions of hybrid additively homomorphic encryption, motivated by its application to hybrid private information retrieval (PIR). Our schemes can be used to combine classical and post-quantum homomorphic encryption to remain secure unless both underlying assumptions fail. Motivated by the generic combiners for key encapsulation mechanisms, digital signatures, and password-authenticated key exchanges, we present parallel and sequential compositions for combining two symmetric additively homomorphic encryption schemes. We provide a generic formal definition of additively homomorphic encryption in which a ciphertext is split into a part containing only randomness and a part that encodes the message. We use this new definition to generalize SimplePIR and FrodoPIR beyond (R)LWE-based cryptosystems, allowing them to achieve hybrid security.

Keywords: Additively homomorphic encryption · Hybrid encryption · Private information retrieval · Post-quantum cryptography.

This is an initial version of the paper, which does not yet include results and is missing security proofs. Many formal definitions are also still missing. We expect to share a full version in the future.

1 Introduction

In the shift to deploying post-quantum cryptography, organizations are opting for hybrids: cryptographic primitives that are proven to be secure under either a classical hardness assumption or one that is conjectured to resist quantum attacks. This means that the primitive does not break if the post-quantum hardness assumption turns out not to be strong enough, or if the classical assumption is broken by a quantum computer.

Some hybrid constructions already exist for primitives such as key encapsulation mechanisms (KEMs) [5] and digital signatures [4]. On a high-level, any two KEMs can be combined into a hybrid KEM by using both in parallel, deriving the final key from a combination of both generated keys. Signatures, similarly, can be composed in parallel to construct a hybrid signature; the verifier must simply ensure that both signatures are valid. Some more complex combiners for KEMs or digital signatures achieve slightly stronger security properties.

Whereas these simpler primitives such as KEMs and signatures permit simple generic constructions, more complex hybrid primitives can be non-trivial to achieve. For example, recently, generic constructions were proposed for password-authenticated key exchanges (PAKEs) [9, 10]. Parallel composition is in those cases only possible when the constituent PAKEs satisfy very strong notions of security. In practice, hybrid PAKEs may use a form of sequential composition that requires weaker security notions.

To the best of our knowledge, there is no literature that considers generic constructions of hybrid homomorphic encryption. A reason for this may be that the plaintext algebra of classical and post-quantum homomorphic encryption schemes is typically wildly different. In this work, we propose the first such constructions, specifically aimed at realizing additively (or linearly) homomorphic encryption. We focus on the symmetric setting, but we note that the main ideas behind these constructions can also be applied to the asymmetric (public-key) setting. Similar to hybrid PAKEs, our constructions include both parallel and sequential compositions.

A strong motivation for the design of these hybrid homomorphic encryption schemes is that they can be used to instantiate private information retrieval (PIR) protocols. Such protocols allow a user to private query a public server without having to download the entire database, and they are already being used in practice [2]. Unfortunately, we are only aware of protocols that are either secure under classical assumptions or they are conjectured to resist quantum attacks under more recent hardness assumptions, which have received less scrutiny. Our hybrid additively homomorphic encryption schemes imply hybrid private information retrieval, which only fails when both hardness assumptions fail.

2 Redefining symmetric AHE

A symmmetric additively homomorphic encryption (AHE) scheme is a cryptosystem with a single key for encryption and decryption and whose ciphertexts can be manipulated in a way that allows performing additions on the underlying plaintexts. We shall use the term encryption scheme and cryptosystem interchangeably. In this section, we provide a slightly different definition of symmetric AHE than the conventional one, in which we separate a ciphertext into a part that only contributes randomness, and a part that carries information about the message. We do so because, later in this work, we compose multiple additively homomorphic cryptosystems into hybrid cryptosystems. This composition is made more efficient by specifying the parts of the ciphertexts that are nothing more than public randomness.

2.1 The cryptosystem

Symmetric AHE schemes are parameterized as follows:

- $\mathcal{K}$: The secret key space

- $\mathcal{M}$: The message (plaintext) space
- $\mathcal{R}$: The public randomness space
- $\mathcal{C}$: The ciphertext space

We require that $\mathcal{M}$ is endowed with a commutative group structure, which can be viewed as a $\mathbb{Z}$-module. The AHE scheme also defines the following interfaces:

- $\mathsf{KeyGen}(1^\kappa) \to sk \in \mathcal{K}$: Generates a secret key with security parameter κ.
- $\mathsf{Encrypt}(sk, \theta, m) \to c \in \mathcal{C}$: Encrypts a message $m \in \mathcal{M}$ using secret key sk and random seed $\theta \in \{0,1\}^*$.
- $\mathsf{Expand}(\theta) \to r \in \mathcal{R}$: Deterministically maps random seed $\theta \in \{0,1\}^*$ to $\mathcal{R}$.
- $\mathsf{Decrypt}(sk, r', c') \to m' \in \mathcal{M}$: Decrypts the randomness-ciphertext pair $(r', c') \in \mathcal{R} \times \mathcal{C}$ using the secret key $sk \in \mathcal{K}$ to produce the plaintext.

Such a cryptosystem is typically defined to be secure if an attacker cannot feasibly extract information about the underlying plaintexts, even when performing chosen-plaintext attacks. This is typically formalized by the symmetric IND-CPA security game [7]. Since we separate r and c, our interface to the security game is slightly different. One may define the conventional encryption-decryption interface as follows, where $\theta \in_R \{0,1\}^\lambda$:

$$\begin{aligned} \mathsf{Encrypt}^*(sk, m) &= (\mathsf{Expand}(\theta), \mathsf{Encrypt}(sk, \theta, m)) \ , \\ \mathsf{Decrypt}^*(sk, (r', c')) &= \mathsf{Decrypt}(sk, r', c') \ . \end{aligned}$$

We also require that the cryptosystem remains IND-CPA secure when the same θ is used to encrypt multiple times but under different keys sk.

2.2 The homomorphic property

It is customary for additively homomorphic cryptosystems to define different functions for performing homomorphic additions between a ciphertext & plaintext and two ciphertexts, as well as performing homomorphic multiplication with a scalar (e.g. using repeated addition). However, with the goal of achieving private information retrieval in mind, we summarize all these operations under a single function that computes a linear operation on multiple ciphertexts. We refer to this operation as a homomorphic dot product and define it below:

- $\mathsf{Dot}(R, C, S) \to (r', c') \in \mathcal{R} \times \mathcal{C}$: Computes a randomness-ciphertext pair that encrypts the linear combination $S \in \mathbb{Z}^n$ of the plaintexts represented by the inputs pairs $R \in \mathcal{R}^n$ and $C \in \mathcal{C}^n$.

For simplicity, this dot product does not allow one to add a plaintext to a ciphertext without encrypting it, as we do not require this functionality for private information retrieval.

It turns out that all the AHE schemes we are aware of, can define the homomorphic dot product in a way that separates randomness R and ciphertexts C. This property allows us to achieve significant performance improvements in

private information retrieval protocols. We say that an AHE scheme is *separable* if it defines the homomorphic dot product as:

$$\mathsf{Dot}(R, C, S) = (\mathsf{Dot}_{\mathcal{R}}(R, S), \mathsf{Dot}_{\mathcal{C}}(C, S)) \ .$$

Some AHE schemes are noisy, and homomorphic operations typically increase this noise. If the noise exceeds a certain threshold, decrypting the ciphertext may result in an incorrect plaintext. As such, we define a relaxed notion of correctness, which states that the probability with which the ciphertext-randomness pair computed by a circuit $\mathcal{C}$ decrypts correctly with overwhelming probability. This circuit $\mathcal{C}$ may be composed of several homomorphic dot products.

2.3 Additional properties

The construction we define later sometimes require specific additional properties to hold for an AHE scheme. We briefly define an additional property that some schemes satisfy: We say that a cryptosystem is *sampleable* if, given sk, an attacker cannot differentiate between $c \in_R \mathcal{C}$ and $\mathsf{Encrypt}(sk, \theta, m)$ where $\theta \in_R \{0,1\}^\lambda$, $r \leftarrow \mathsf{Expand}(\theta)$, and $m \leftarrow \mathsf{Decrypt}(sk, r, c)$. This is therefore a strictly stronger property than the uniformity of ciphertexts.

We also define a property that does not relate to security, but concerns the homomorphic operation. We say that an additively homomorphic cryptosystem is *algebraic* if the homomorphic dot product is defined as follows:

$$\mathsf{Dot}_{\mathcal{R}}(R, S) = R \cdot S \ ,$$
$$\mathsf{Dot}_{\mathcal{C}}(C, S) = C \cdot S \ .$$

In other words, the homomorphic dot product is simply defined as a regular dot product (e.g. using repeated addition) on $\mathcal{R}$ and $\mathcal{C}$. An algebraic AHE scheme is by definition always separable.

3 Additively homomorphic secret sharing

Our parallel compositions require a specific form of secret sharing.

3.1 Generic definition

Our parallel composition is parameterized by an additively homomorphic secret sharing scheme (AHSS), which we define below. For simplicity, we only consider the case where secrets are split into two shares. The secret sharing schemes is parameterized by the following variables:

- $\mathcal{P}$: The space of secrets that may be shared
- $\mathcal{S}_1$: The space of the first secret share
- $\mathcal{S}_2$: The space of the second secret share

Notice that unlike in typical secret sharing schemes, the two shares may now be elements of different algebras. The secret sharing scheme also defines the following interfaces:

- $(s_1, s_2) \leftarrow \mathsf{Split}(p)$: Splits secret plaintext $p \in \mathcal{P}$ into two shares $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$.
- $(s_1, s_2) \leftarrow \mathsf{Combine}(s_1, s_2,)$: Splits secret plaintext $p \in \mathcal{P}$ into two shares $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$.

Given $(s_1, s_2) \leftarrow \mathsf{Split}(p)$ and $(s'_1, s'_2) \leftarrow \mathsf{Split}(p')$, we must have that:

$$\mathsf{Combine}(s_1 + s'_1, s_2 + s'_2) = p + p' \tag{1}$$

This defines the additively homomorphic property of the secret sharing scheme. The secret sharing scheme must be information-theoretically secure in the sense that knowledge of only one of the shares does not allow an adversary to gain a non-negligible amount of information about the secret.

3.2 Sampleable AHSS

If $\mathcal{S}_1$ and $\mathcal{S}_2$ are isomorphic, then we can simply use regular additive secret sharing. The resulting scheme is also sampleable in the sense that you can sample $s_1 \in_R \mathcal{S}_1$ and still come up with a valid and secure $s_2 \in \mathcal{S}_2$ that, in isolation, information-theoretically hides any specific secret $\mathcal{P}$. For this purpose, we define the following function:

- $s_2 \leftarrow \mathsf{SplitSampled}(p, s_1)$: Splits secret $p \in \mathcal{P}$ into previously sampled $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$.

3.3 Small-value AHSS

If $\mathcal{S}_1$ and $\mathcal{S}_2$ are not isomorphic, they may not even have the same cardinality, we can still come up with a valid secret sharing scheme. Say that $\mathcal{P} = \mathbb{Z}_P$, $\mathcal{S}_1 = \mathbb{Z}_{S_1}$, and $\mathcal{S}_2 = \mathbb{Z}_{S_2}$, with $P \ll S_1$ and $P \ll S_2$. We can split a secret using regular additive secret sharing and embed the shares into S_1 and S_2. As long as we do not perform too many homomorphic operations on the secret shares, the embedded shares do not wrap around the modulus, so they remain correct. We can combine the shares by reducing them modulo P before adding the embedded shares together.

4 Constructions for hybrid symmetric AHE

We now discuss four different compositions to achieve hybrid AHE.

4.1 Parallel composition: One-&-Two

Given two AHE schemes AHE_1 and AHE_2, we define the composition $\mathsf{AHE}_{1\&2}$ as follows. It is parameterized as follows:

- AHSS: The additively homomorphic secret sharing scheme.
- AHE_1: The first AHE scheme, with no additional properties.
- AHE_2: The second AHE scheme, with no additional properties.

For an AHSS scheme to be suitable, we require that the following constraints are met:

$$\mathsf{AHSS}.\mathcal{S}_1 = \mathsf{AHE}_1.\mathcal{M}$$
$$\mathsf{AHSS}.\mathcal{S}_2 = \mathsf{AHE}_2.\mathcal{M}$$

We can then define the rest of our parallel composition:

$$\begin{aligned}
\mathsf{AHE}_{1\&2}.\mathcal{K} &= \mathsf{AHE}_1.\mathcal{K} \times \mathsf{AHE}_2.\mathcal{K} \\
\mathsf{AHE}_{1\&2}.\mathcal{M} &= \mathsf{AHSS}.\mathcal{P} \\
\mathsf{AHE}_{1\&2}.\mathcal{R} &= \mathsf{AHE}_1.\mathcal{R} \times \mathsf{AHE}_2.\mathcal{R} \\
\mathsf{AHE}_{1\&2}.\mathcal{C} &= \mathsf{AHE}_1.\mathcal{C} \times \mathsf{AHE}_2.\mathcal{C} \\
\mathsf{AHE}_{1\&2}.\mathsf{KeyGen}(1^\kappa) &= (\mathsf{AHE}_1.\mathsf{KeyGen}(1^\kappa), \mathsf{AHE}_2.\mathsf{KeyGen}(1^\kappa)) \\
\mathsf{AHE}_{1\&2}.\mathsf{Expand}(\theta) &= (\mathsf{AHE}_1.\mathsf{Expand}(\theta), \mathsf{AHE}_2.\mathsf{Expand}(\theta)) \\
\mathsf{AHE}_{1\&2}.\mathsf{Dot}_{\mathcal{R}}((R_1, R_2), S) &= (\mathsf{AHE}_1.\mathsf{Dot}_{\mathcal{R}}(R_1, S), \mathsf{AHE}_2.\mathsf{Dot}_{\mathcal{R}}(R_2, S)) \\
\mathsf{AHE}_{1\&2}.\mathsf{Dot}_{\mathcal{C}}((C_1, C_2), S) &= (\mathsf{AHE}_1.\mathsf{Dot}_{\mathcal{C}}(C_1, S), \mathsf{AHE}_2.\mathsf{Dot}_{\mathcal{C}}(C_2, S))
\end{aligned}$$

We define $\mathsf{AHE}_{1\&2}.\mathsf{Encrypt}((sk_1, sk_2), \theta, m)$ as:

- Split the message: $(s_1, s_2) \leftarrow \mathsf{AHSS}.\mathsf{Split}(m)$.
- Output $(\mathsf{AHE}_1.\mathsf{Encrypt}(sk_1, \theta, s_1), \mathsf{AHE}_2.\mathsf{Encrypt}(sk_2, \theta, s_2))$.

and we define $\mathsf{AHE}_{1\&2}.\mathsf{Decrypt}((sk_1, sk_2), (r_1, r_2), (c_1, c_2))$ as:

- Decrypt the first share: $s_1 \leftarrow \mathsf{AHE}_1.\mathsf{Decrypt}(sk_1, r_1, c_1)$.
- Decrypt the second share: $s_2 \leftarrow \mathsf{AHE}_2.\mathsf{Decrypt}(sk_2, r_2, c_2)$.
- Output $\mathsf{AHSS}.\mathsf{Combine}(s_1, s_2)$.

4.2 Parallel composition with sampling: Sampled-One-&-Two

The above construction does not require the AHE schemes to satisfy special properties, but it turns out we can improve its efficiency if we are able to assume that one of the schemes is sampleable. We assume it is AHE_1 that is sampleable without loss of generality. The idea is to sample the first ciphertext from a seed, then decrypt it and interpret it as a secret share, and finally to base the second share on this share so it reconstructs to the message we wish to encrypt when combined. We describe what adjustments to make below.

For an AHSS scheme to be suitable, we require the additional constraint that it is sampleable as well. We can then redefine some parts of this composition:

$$\begin{aligned}
\mathsf{AHE}_{s1\&2}.\mathcal{R} &= \mathsf{AHE}_1.\mathcal{R} \times \mathsf{AHE}_2.\mathcal{R} \times \mathsf{AHE}_1.\mathcal{C} \\
\mathsf{AHE}_{s1\&2}.\mathcal{C} &= \mathsf{AHE}_2.\mathcal{C} \\
\mathsf{AHE}_{s1\&2}.\mathsf{Expand}(\theta) &= (\mathsf{AHE}_1.\mathsf{Expand}(\theta), \mathsf{AHE}_2.\mathsf{Expand}(\theta), H(\theta)) \\
\mathsf{AHE}_{s1\&2}.\mathsf{Dot}_{\mathcal{R}}((R_1, R_2, C_1), S) &= (\mathsf{AHE}_1.\mathsf{Dot}_{\mathcal{R}}(R_1, S), \mathsf{AHE}_2.\mathsf{Dot}_{\mathcal{R}}(R_2, S), \mathsf{AHE}_1.\mathsf{Dot}_{\mathcal{C}}(C_1, S)) \\
\mathsf{AHE}_{s1\&2}.\mathsf{Dot}_{\mathcal{C}}(C_2, S) &= \mathsf{AHE}_2.\mathsf{Dot}_{\mathcal{C}}(C_2, S)
\end{aligned}$$

Here, H is a pseudo-random function that maps $\theta \in \{0,1\}^\lambda$ to $\mathsf{AHE}_1.\mathcal{C}$.
We define $\mathsf{AHE}_{s1\&2}.\mathsf{Encrypt}((sk_1, sk_2), \theta, m)$ as:

- Sample the first ciphertext: $c_1 \leftarrow H(\theta)$.
- Determine the first share: $s_1 \leftarrow \mathsf{AHE}_1.\mathsf{Decrypt}(sk_1, \mathsf{AHE}_1.\mathsf{Expand}(\theta), c_1)$.
- Split the message: $s_2 \leftarrow \mathsf{AHSS}.\mathsf{SplitSampled}(m, s_1)$.
- Output $\mathsf{AHE}_2.\mathsf{Encrypt}(sk_2, \theta, s_2)$.

and we define $\mathsf{AHE}_{s1\&2}.\mathsf{Decrypt}((sk_1, sk_2), (r_1, r_2, c_1), c_2)$ as:

- Decrypt the first share: $s_1 \leftarrow \mathsf{AHE}_1.\mathsf{Decrypt}(sk_1, r_1, c_1)$.
- Decrypt the second share: $s_2 \leftarrow \mathsf{AHE}_2.\mathsf{Decrypt}(sk_2, r_2, c_2)$.
- Output $\mathsf{AHSS}.\mathsf{Combine}(s_1, s_2)$.

4.3 Sequential composition: One-encrypts-Two

In the parallel compositions described above, the AHE schemes encrypt secret shares. We now consider sequential compositions, in which one AHE scheme encrypts the ciphertext of another AHE scheme. A benefit of this approach is that you do not have to store both ciphertexts.

In this first sequential construction, there is an inner and outer AHE scheme. We require that the inner AHE scheme is algebraic, which ensures that homomorphic addition on the outer scheme also implies homomorphic addition on the inner one. One problem with having one scheme encrypt the ciphertexts produced by another is that many combinations of AHE schemes are incompatible: the plaintext space of one does not match the ciphertext space of the other. We circumvent this problem by parameterizing our construction with a mapping that provides a translation between these two spaces. To ensure that homomorphic additions on the outer scheme still cause homomorphic additions in the inner one, the mapping must be a group isomorphism.

Given two AHE schemes, AHE_1 and AHE_2, the latter being algebraic, we formally define the composition AHE_{1e2} as follows. It has the following parameters:

- AHE_1: The first AHE scheme, with no additional properties.
- AHE_2: The second AHE scheme, which must be algebraic.
- Iso: A group isomorphism from $\mathsf{AHE}_2.\mathcal{C}$ to $\mathsf{AHE}_1.\mathcal{M}$.

The rest of the scheme is defined as follows:

$$\begin{aligned}
\mathsf{AHE}_{1e2}.\mathcal{K} &= \mathsf{AHE}_1.\mathcal{K} \times \mathsf{AHE}_2.\mathcal{K} \\
\mathsf{AHE}_{1e2}.\mathcal{M} &= \mathsf{AHE}_2.\mathcal{M} \\
\mathsf{AHE}_{1e2}.\mathcal{R} &= \mathsf{AHE}_1.\mathcal{R} \times \mathsf{AHE}_2.\mathcal{R} \\
\mathsf{AHE}_{1e2}.\mathcal{C} &= \mathsf{AHE}_1.\mathcal{C} \\
\mathsf{AHE}_{1e2}.\mathsf{KeyGen}(1^\kappa) &= (\mathsf{AHE}_1.\mathsf{KeyGen}(1^\kappa), \mathsf{AHE}_2.\mathsf{KeyGen}(1^\kappa)) \\
\mathsf{AHE}_{1e2}.\mathsf{Expand}(\theta) &= (\mathsf{AHE}_1.\mathsf{Expand}(\theta), \mathsf{AHE}_2.\mathsf{Expand}(\theta)) \\
\mathsf{AHE}_{1e2}.\mathsf{Encrypt}((sk_1, sk_2), \theta, m) &= \mathsf{AHE}_1.\mathsf{Encrypt}(sk_1, \theta, \mathsf{Iso}(\mathsf{AHE}_2.\mathsf{Encrypt}(sk_2, \theta, m))) \\
\mathsf{AHE}_{1e2}.\mathsf{Decrypt}((sk_1, sk_2), (r_1, r_2), c) &= \mathsf{AHE}_2.\mathsf{Decrypt}(sk_2, r_2, \mathsf{Iso}^{-1}(\mathsf{AHE}_1.\mathsf{Decrypt}(sk_1, r_1, c))) \\
\mathsf{AHE}_{1e2}.\mathsf{Dot}_\mathcal{R}((R_1, R_2), S) &= (\mathsf{AHE}_1.\mathsf{Dot}_\mathcal{R}(R_1, S), \mathsf{AHE}_2.\mathsf{Dot}_\mathcal{R}(R_2, S)) \\
\mathsf{AHE}_{1e2}.\mathsf{Dot}_\mathcal{C}((C_1, C_2), S) &= \mathsf{AHE}_1.\mathsf{Dot}_\mathcal{C}(C_1, S)
\end{aligned}$$

4.4 Sequential composition with sampling: One-masks-Two

In our parallel compositions, we showed that sampling allowed us to omit one of the AHE scheme's ciphertexts from the composed ciphertext space. It turns out that a similar trick works for our sequential composition. The idea is to sample a random ciphertext from the first AHE scheme and to use its plaintext to additively mask the ciphertext from the second scheme. We can then include first ciphertext in the randomness space so that the ciphertext space only contains the second ciphertext.

Like above, we can improve the compatibility of composed schemes by introducing a mapping. In this case, that mapping suffices to be one-way. After all, we only need to undo the mask. As such, the mapping suffices to be a group homomorphism.

Given two AHE schemes, AHE_1 and AHE_2, the latter being algebraic, we formally define the composition AHE_{1e2} as follows. It has the following parameters:

- AHE_1: The first AHE scheme, which must be sampleable.
- AHE_2: The second AHE scheme, which must be algebraic.
- Hom: A group homomorphism from $\mathsf{AHE}_1.\mathcal{M}$ to $\mathsf{AHE}_2.\mathcal{C}$.

The rest of the scheme is defined as follows:

$$\begin{aligned}
\mathsf{AHE}_{1e2}.\mathcal{K} &= \mathsf{AHE}_1.\mathcal{K} \times \mathsf{AHE}_2.\mathcal{K} \\
\mathsf{AHE}_{1e2}.\mathcal{M} &= \mathsf{AHE}_2.\mathcal{M} \\
\mathsf{AHE}_{1e2}.\mathcal{R} &= \mathsf{AHE}_1.\mathcal{R} \times \mathsf{AHE}_2.\mathcal{R} \times \mathsf{AHE}_1.\mathcal{C} \\
\mathsf{AHE}_{1e2}.\mathcal{C} &= \mathsf{AHE}_2.\mathcal{C} \\
\mathsf{AHE}_{1e2}.\mathsf{KeyGen}(1^\kappa) &= (\mathsf{AHE}_1.\mathsf{KeyGen}(1^\kappa), \mathsf{AHE}_2.\mathsf{KeyGen}(1^\kappa)) \\
\mathsf{AHE}_{1e2}.\mathsf{Expand}(\theta) &= (\mathsf{AHE}_1.\mathsf{Expand}(\theta), \mathsf{AHE}_2.\mathsf{Expand}(\theta), H(\theta)) \\
\mathsf{AHE}_{1e2}.\mathsf{Dot}_\mathcal{R}((R_1, R_2, C_1), S) &= (\mathsf{AHE}_1.\mathsf{Dot}_\mathcal{R}(R_1, S), \mathsf{AHE}_2.\mathsf{Dot}_\mathcal{R}(R_2, S), \mathsf{AHE}_1.\mathsf{Dot}_\mathcal{C}(C_1, S)) \\
\mathsf{AHE}_{1e2}.\mathsf{Dot}_\mathcal{C}(C_2, S) &= \mathsf{AHE}_2.\mathsf{Dot}_\mathcal{C}(C_2, S)
\end{aligned}$$

Here, H is a pseudo-random function that maps $\theta \in \{0,1\}^\lambda$ to $\mathsf{AHE}_1.\mathcal{C}$.
We define $\mathsf{AHE}_{s1\&2}.\mathsf{Encrypt}((sk_1, sk_2), \theta, m)$ as:

- Sample the first ciphertext: $c_1 \leftarrow H(\theta)$.
- Determine the mask: $\mu \leftarrow \mathsf{AHE}_1.\mathsf{Decrypt}(sk_1, \mathsf{AHE}_1.\mathsf{Expand}(\theta), c_1)$.
- Encrypt the message: $c_2 \leftarrow \mathsf{AHE}_2.\mathsf{Encrypt}(sk_2, \theta, s_2)$.
- Output the masked ciphertext $c_2 + \mu$.

and we define $\mathsf{AHE}_{s1\&2}.\mathsf{Decrypt}((sk_1, sk_2), (r_1, r_2, c_1), c_2)$ as:

- Decrypt the mask: $\mu \leftarrow \mathsf{AHE}_1.\mathsf{Decrypt}(sk_1, r_1, c_1)$.
- Decrypt the masked ciphertext: $c_2' \leftarrow \mathsf{AHE}_2.\mathsf{Decrypt}(sk_2, r_2, c_2)$.
- Output $c_2' - \mu$.

5 Efficient instantiations of hybrid AHE

We discuss how to practically parameterize our hybrid AHE constructions.

5.1 Practical realizations of AHE

In Table 1 we list the two classical and two post-quantum AHE schemes we consider in this work. Using the four generic constructions described in Section 4, we can construct at least 16 different hybrid AHE schemes.

Table 1. The practical constructions for AHE that we consider for realizing hybrid AHE. The classical schemes rely on the decisional Diffie-Hellman or the decisional composite residuosity assumptions, whereas the post-quantum schemes rely on (a ring variant of) the learning with errors assumption. All schemes are seperable. The post-quantum schemes are not sampleable. Paillier is not algebraic because its homomorphic addition is realized using regular multiplication.

AHE scheme	Security	Assumption	Sep.	Sampl.	Alg.
$\mathbb{G}$-ElGamal	Classical	DDH	●	●	●
Paillier		DCR	●	●	○
Regev	Post-quantum	LWE	●	○	●
BFV		RLWE	●	○	●

5.2 Parameter selection

Parameterizing ElGamal We choose to parameterize $\mathbb{G}$-ElGamal using a prime-order elliptic curve (encoding). Any such curve would work, but we chose Curve25519 with the Ristretto encoding as it permits a mapping between bit-strings and curve points. This allows us to encode the values of a PIR database.

It is conjectured that this group is secure under the DDH assumption. ElGamal's homomorphic operations remain correct for any circuit. A downside of this cryptosystem is that decryption requires computing an intractable discrete logarithm. Fortunately, if we limit the ciphertexts we want to decrypt to those whose plaintext is in a small range, we can create a lookup table, which allows us to decrypt efficiently after a one-time setup.

Parameterizing Paillier For Paillier, we use the key length recommendations provided by NIST for RSA [3], which states that the RSA modulus must have length at least 3072 bits to achieve 128 bits of security. Note that the modulus of a Paillier ciphertext is the square of the RSA modulus, so the actual size of such a ciphertext will be at least 6144 bits. Like ElGamal, Paillier remains correct for any circuit.

It is straightforward to map bitstrings to and from Paillier plaintexts, as the plaintexts are integers under the RSA modulus. This means that we can encode and decode 3072-bit bitstrings by interpreting them as integers.

Parameterizing the (R)LWE-based cryptosystems For the Regev and the BFV cryptosystem, we propose to follow the homomorphic encryption standard [1], which contains tables describing the maximal permitted ciphertext modulus bitlength that remains secure under quantum attacks for a fixed noise magnitude. If a large plaintext modulus is required, we must choose a large ciphertext modulus and therefore a large dimension. If a large ciphertext modulus is required, we can simply scale up a smaller sample. Whether a certain parameter set is valid depends on the aspects of the PIR protocol we want to run: if the database is large, the ciphertext must have a higher noise thrshold, and if the database's values are large, we require a larger plaintext modulus.

5.3 Parameterizing the constructions

While the constructions proposed in Section 4 have few constraints, they do require that the plaintext and ciphertext spaces of the composed AHE schemes are somehow compatible. The classical schemes involve plaintext and ciphertext spaces with large orders (thousands of bits) and few factors. These factors are either very small (e.g. the cofactor of an elliptic curve group) or very large (e.g. the prime factors of an RSA modulus). On the other hand, learning with errors-based post-quantum AHE schemes prefer ciphertext moduli that decompose into multiple word-sized primes, and the plaintext spaces are often significantly smaller than the classical ones. Below, we will discuss how to parameterize the constructions when instantiated with the AHE schemes from Table 1, and we analyze how many plaintext bits they can encrypt for each ciphertext bit.

The key idea to unify these two different types of AHE schemes is to select the (R)LWE ciphertext moduli so they match the plaintext or ciphertext space of ElGamal or Paillier. For ElGamal, this is the group of integers modulo a specific 252-bit prime modulus. For Paillier, the modulus is 3072 bits in size. To

make sure the resultant ciphertexts are not impractically large, we can set the actual ciphertext modulus to be smaller, and scale it up simply by multiplying it using a predetermined factor. Of course, this also scales up the noise, so the ciphertext cannot be arbitrary small. This scaling must be done in the AHSS scheme, the Iso mapping, or the Hom mapping.

6 Hybrid private information retrieval

In this section, we provide a generic construction for private information retrieval from separable symmetric AHE schemes. By instantiating the AHE scheme with one that provides hybrid security, the PIR protocol achieves the same. Our construction is a straightforward generalization of SimplePIR [8] and FrodoPIR [6]. The resulting PIR protocol involves only one server and requires no client-specific storage. It does require each client to download a public hint.

The PIR protocol is parameterized by a separable AHE scheme $\mathsf{AHE}_{\mathsf{sep}}$. Instead of as rows that exploit the fact that an RLWE-based ciphertext can encode multiple elements, we just encode larger values from which the client can eventually select their queried sub-value. We also do not constrain the size of the database. We use the same syntax as SimplePIR.

$\mathsf{Setup}(\mathsf{db}) \to (\mathsf{hint}_s, \mathsf{hint}_c)$:

- Sample $\theta \in_R \{0,1\}^\lambda$.
- Compute $r \leftarrow \mathsf{Expand}(\theta)$.
- Return $(\perp, (\theta, \mathsf{AHE}_{\mathsf{sep}}.\mathsf{Dot}_{\mathcal{R}}(r, \mathsf{db})))$.

$\mathsf{Query}(i) \to (\mathsf{state}, \mathsf{query})$:

- Compute one-hot encoding $u \leftarrow \mathsf{OneHot}(i, N)$.
- Create a fresh key $sk \leftarrow \mathsf{AHE}_{\mathsf{sep}}.\mathsf{KeyGen}(1^\kappa)$.
- Create encryption(s): $c \leftarrow \mathsf{AHE}_{\mathsf{sep}}.\mathsf{Encrypt}(sk, \theta, u)$.
- Return (sk, c).

$\mathsf{Answer}(\mathsf{db}, \mathsf{hint}_s, \mathsf{query}) \to \mathsf{answer}$:

- Return $\mathsf{AHE}_{\mathsf{sep}}.\mathsf{Dot}_{\mathcal{C}}(\mathsf{query}, \mathsf{db}))$.

$\mathsf{Recover}(\mathsf{state}, \mathsf{hint}_c, \mathsf{answer}) \to VALUE$:

- Return $\mathsf{AHE}_{\mathsf{sep}}.\mathsf{Decrypt}(\mathsf{state}, \mathsf{hint}_c, \mathsf{answer}))$.

We will provide results in the full version.

References

1. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption standard. Technical Report v1.1, HomomorphicEncryption.org (Nov 21 2018), https://homomorphicencryption.org/wp-content/uploads/2018/11/HomomorphicEncryptionStandardv1.1.pdf

2. Apple: Combining machine learning and homomorphic encryption in the apple ecosystem. https://machinelearning.apple.com/research/homomorphic-encryption (October 24 2024), accessed: 2025-08-29
3. Barker, E.B.: Recommendation for key management: Part 1 – general. Tech. Rep. NIST Special Publication (SP) 800-57 Part 1, Rev. 5, National Institute of Standards and Technology, Gaithersburg, MD (May 2020). https://doi.org/10.6028/NIST.SP.800-57pt1r5, https://doi.org/10.6028/NIST.SP.800-57pt1r5
4. Bindel, N., Hale, B.: A note on hybrid signature schemes. Cryptology ePrint Archive, Paper 2023/423 (2023), https://eprint.iacr.org/2023/423
5. Connolly, D., Hövelmanns, K., Hülsing, A., Kousidis, S., Meijers, M.: Starfighters — on the general applicability of x-wing. Cryptology ePrint Archive, Paper 2025/1397 (2025), https://eprint.iacr.org/2025/1397
6. Davidson, A., Pestana, G., Celi, S.: FrodoPIR: Simple, scalable, single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/981 (2022), https://eprint.iacr.org/2022/981
7. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences **28**(2), 270–299 (Apr 1984)
8. Henzinger, A., Hong, M.M., Corrigan-Gibbs, H., Meiklejohn, S., Vaikuntanathan, V.: One server for the price of two: Simple and fast single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/949 (2022), https://eprint.iacr.org/2022/949
9. Hesse, J., Rosenberg, M.: PAKE combiners and efficient post-quantum instantiations. Cryptology ePrint Archive, Paper 2024/1621 (2024), https://eprint.iacr.org/2024/1621
10. Lyu, Y., Liu, S.: Hybrid password authentication key exchange in the UC framework. Cryptology ePrint Archive, Paper 2024/1630 (2024), https://eprint.iacr.org/2024/1630