Interpolating Predicate and Functional Encryption from Learning With Errors

Shweta Agrawal *

Abstract

We construct a functional encryption scheme for circuits which achieves a notion of security that interpolates predicate and functional encryption. Our scheme is secure based on the subexponential learning with errors (LWE) assumption. Our construction simultaneously achieves and improves upon the security of the current best known, and incomparable, constructions from standard assumptions, namely the predicate encryption scheme of Gorbunov, Vaikuntanathan and Wee (CRYPTO 2015) and the reusable garbled circuits scheme of Goldwasser, Kalai, Popa, Vaikuntanathan and Zeldovich (STOC 2013). Our contributions may be summarized as follows.

1. We show that existing LWE based predicate encryption schemes [AFV11, GVW15] are completely insecure against a general functional encryption adversary (i.e. in the "strong attribute hiding" game). We demonstrate three different attacks, the strongest of which is applicable even to the inner product predicate encryption scheme [AFV11]. Our attacks are practical and allow the attacker to completely recover $\mathbf{a}$ from its encryption $\mathsf{Enc}(\mathbf{a})$ within a polynomial number of queries. This illustrates that the barrier between predicate and functional encryption is not just a limitation of proof techniques.
2. We provide a new construction that unifies and extends the constructions of Gorbunov et al. [GVW15] and Goldwasser et al. [GTKP+13]. Our construction supports a single "decryption query" as in [GTKP+13] in addition to an unbounded number of "non-decryption queries" as in [GVW15]. In particular, our construction yields an alternate candidate for reusable garbled circuits.
3. We upgrade the security of our construction, as well as [AFV11, GVW15], from selective to *semi-adaptive*, where the adversary may output the challenge *after* seeing the public parameters in the security game. Our transformation is generic, and applies to several LWE based selectively secure FE schemes.
4. We generalise the above scheme to support q decryption queries, for any polynomial q which is a-priori fixed. The ciphertext size grows as $O(q^2)$, improving upon the q-query version of [GTKP+13, GVW12] in which the ciphertext size grows as $O(q^4)$. Our ciphertext is succinct in that its size does not depend on the size of the circuit.

*IIT Madras, India. Email: shweta@iitm.ac.in

1 Introduction

The last decade has witnessed important progress in the field of computing on encrypted data. Several sophisticated generalizations of encryption, such as Identity Based Encryption [BF01, Coc01, GPV08], Attribute Based Encryption [GPSW06, BSW07, GGH+13c, GVW13], Predicate Encryption [KSW08, AFV11, GVW15], Fully Homomorphic Encryption [Gen09, BV11, GSW13, BV14], Property Preserving Encryption [PR12] have burst onto the scene, significantly advancing the capabilities of modern cryptography.

These generalizations aim to provide the capability of computing "blind-folded" – namely, given an encryption of some data $\mathbf{a}$, an untrusted party should be able to perform computations on $\mathsf{Enc}(\mathbf{a})$ so that the resultant ciphertext may be decrypted meaningfully. The notion of fully homomorphic encryption permits arbitrary computation on encrypted data, but restricts decryption to be all-or-nothing, namely, the holder of the secret key may decrypt the resultant ciphertext to learn the result of the computation, but the same key also decrypts the original ciphertext revealing $\mathbf{a}$. For applications that require restricted access to results of the computation, the notion of functional encryption (FE) is more suitable. In functional encryption, a secret key is associated with a function, typically represented as a circuit C, denoted by SK_C and a ciphertext with some input $\mathbf{a}$ from the domain of C, denoted by $\mathsf{CT}_{\mathbf{a}}$. Given SK_C and $\mathsf{CT}_{\mathbf{a}}$, the user may run the decryption procedure to learn the value $C(\mathbf{a})$. Security of the system guarantees that nothing beyond $C(\mathbf{a})$ can be learned from $\mathsf{CT}_{\mathbf{a}}$ and SK_C. Functional encryption was formalized by Boneh et al. [BSW11] to unify and extend the notions of Identity Based Encryption, Attribute Based Encryption and Predicate Encryption, which had already appeared in the literature.

There has been considerable progress in the last several years towards constructing FE for advanced functionalities [BF01, Coc01, BW06, BW07, GPV08, CHKP10, ABB10, GPSW06, BSW07, KSW08, LOS+10, AFV11, Wat12, GVW13, GGH+13c, GGH+13b, GVW15]. For the most powerful notion of "full-fledged" functional encryption, that allows the evaluation of arbitrary efficiently-computable functions and is secure against general adversaries, the only known constructions rely on multilinear maps [GGHZ14] or indistinguishability obfuscation (iO) [GGH+13b]. However, all known candidate multi-linear map constructions [GGH13a, CLT13, GGH15] as well as some candidates of iO have been recently broken [CHL+15, CGH+15, HJ15, CJL, CFL+16, MSZ16].

From standard assumptions, the best known constructions do support general functionalities, but achieve restricted notions of security. Currently, the state-of-the-art comprises two incomparable constructions:

- The reusable garbled circuits construction of Goldwasser et al. [GTKP+13], which supports all polynomial sized Boolean circuits but restricts the attacker to only obtain a single secret key, for any circuit C of her choice. This construction can be compiled with the bounded collusion FE construction of [GVW12] to obtain a scheme which supports q queries, for any a-priori bounded q, and with a ciphertext size that grows as $O(q^4)$. Note that the ciphertext size here does not depend on the size of the circuit C, and is thus *succinct*.

- The recent predicate encryption for circuits construction of Gorbunov et al. [GVW15], which also supports all polynomial sized Boolean circuits but restricts the attacker to only acquire keys for circuits C_i such that $C_i(\mathbf{a}) = 0$, when $\mathbf{a}$ is the vector of challenge attributes. He may not request any keys C_j such that $C_j(\mathbf{a}) = 1$. We will refer to the former as 0-keys and the latter as 1-keys. This restricted game of security is often referred to as *weak attribute hiding* in the literature.

Both constructions natively achieve the restricted *selective* notion of security, which forces the

attacker to output the challenge in the very first step of the game, before seeing the public parameters.

Covering the distance from these restricted constructions to full fledged functional encryption is a much sought-after goal, and one that must contend with several thorny technical issues. The former construction relies on the use of garbled circuits for decryption, which restricts the number of supported keys to 1, or, using the additional machinery of [GVW12], to some a-priori bounded q. The use of garbled circuits is central to this construction, and surmounting the bounded key limitation appears to require entirely new techniques. On the other hand, the second construction does support an unbounded number of queries, but restricts them to belong to the 0-set. It is unclear how to support even a single 1-query in this case, due to various technical hurdles that arise from the proof techniques (more on this below). This motivates the following question:

Can we provide a single construction that supports an unbounded number of 0*-queries and a bounded number of* 1*-queries?*

1.1 Our Contributions

In this work, we answer the above question in the affirmative and provide a construction that simultaneously achieves (and improves upon) the best of both [GVW15] and [GTKP+13]. Our contributions may be summarized as follows.

1. We demonstrate that the inability to handle 1-queries by the proofs of current predicate encryption systems [GVW15, AFV11] is not a limitation of proof technique. We show that these constructions are completely insecure against an adversary who requests an arbitrary number of 1-keys. We demonstrate three different attacks, the strongest of which is even applicable to the simple inner product testing functionality of [AFV11]. Our attacks are practical and allow an attacker making a polynomial number of 1-queries to completely recover $\mathbf{a}$ from $\mathsf{CT}_{\mathbf{a}}$.

2. We provide a new construction that supports a single 1-query, and an unbounded number of 0-queries, unifying the best of both [GVW15, GTKP+13]. We refer to our construction as a $(1, \text{poly})$-FE scheme, where the first term refers to the number of 1 queries, and the second to the number of 0 queries supported.

3. We upgrade the security of our construction, as well as [AFV11, GVW15], from selective to *semi-adaptive* [CW14] where the adversary may output the challenge *after* seeing the public parameters in the security game. Our transformation is generic, and applies to several LWE based selectively secure FE schemes.

4. We generalize our $(1, \text{poly})$-FE to (Q, poly)-FE for any polynomial Q which is a-priori fixed. The ciphertext size grows as $O(Q^2)$, improving upon the Q-query version of [GTKP+13, GVW12] in which the ciphertext size grows as $O(Q^4)$. Our ciphertext is succinct in that its size does not depend on the size of the circuit. However, our security game is weaker than the standard FE bounded collusion game [GVW12, AR16] in that the attacker is forced to ask all the 1 queries before asking any 0 queries.

We provide a table comparing our results with previous work in Figure 1.

1.2 Our Techniques

Our work builds upon the constructions of Goldwasser et al. [GTKP+13] and Gorbunov et al. [GVW15]. Both these systems begin with the idea that the public attributes in an attribute based

	0 keys	1-keys	CT succinct?	CT Q-dependence	Security
[GVW15]	poly	0	Y	—	selective
[GTKP+13] + [GVW12]	Q_1	$Q - Q_1$	Y	Q^4	selective
[AR16]	Q_1	$Q - Q_1$	N	Q^2	full
This	poly	Q	Y	Q^2	semi-adaptive

Figure 1: Comparison of results with prior work.

encryption scheme (ABE) may be hidden, and yet remain amenable to computation, if they are encrypted using fully homomorphic encryption. Recall that in an attribute based encryption scheme [GPSW06], a ciphertext is associated with a public attribute vector $\mathbf{a}$ and plaintext bit μ, and it hides μ, but not $\mathbf{a}$.

Then, to hide $\mathbf{a}$, encrypt it using FHE to obtain $\widehat{\mathbf{a}}$, and treat this encryption as the public attribute in an ABE system. Since an ABE scheme for circuits [GVW13, BGG+14] allows for a key SK_C to evaluate an arbitrary circuit C on the attribute, the decryptor may now homomorphically compute on $\widehat{\mathbf{a}}$ using the FHE evaluation circuit. Then, given a key corresponding to the circuit $\mathsf{FHE.Eval}(\mathsf{C}, \cdot)$, the decryptor may run the ABE decryption procedure to learn the FHE encryption of $C(\mathbf{a})$, namely $\widehat{C(\mathbf{a})}$.

This is not yet enough, as the goal is for the decryptor to learn $C(\mathbf{a})$ in the clear. To achieve this, FHE decryption must be performed on $\widehat{C(\mathbf{a})}$ in a manner that does not permit decryption of any ciphertext other than $\widehat{C(\mathbf{a})}$. The scheme of Goldwasser et al. [GTKP+13] resolves this difficulty by employing a single use garbled circuit for the FHE decryption function and using ABE to provide labels corresponding to input $\widehat{C(\mathbf{a})}$. This constrains FHE decryption, but restricts the resultant FE scheme to only be secure against a single key request. The scheme of Gorbunov et al. [GVW15] resolves this difficulty by making use of two nicely matching asymmetries:

1. *The asymmetry in computation.* To compute $C(\mathbf{a})$ using the above method, the bulk of the computation is to be performed on FHE ciphertext, namely $\mathsf{FHE.Eval}(C, \widehat{\mathbf{a}})$, where $\widehat{\mathbf{a}}$ can be public. The remainder of the computation, namely running the FHE decryption circuit on $\widehat{C(\mathbf{a})}$, is a relatively lightweight circuit.

2. *The asymmetry in attribute hiding in the ABE scheme of [BGG+14].* There is an inherent asymmetry in the homomorphic multiplication procedure of the ABE scheme [BGG+14], so that computing a product of two ciphertexts wth attributes $\mathbf{a}_1$ and $\mathbf{a}_2$ respectively, only necessitates revealing *one* attribute (say $\mathbf{a}_1$) while the other ($\mathbf{a}_2$) can remain hidden (for addition, both $\mathbf{a}_1$ and $\mathbf{a}_2$ may remain hidden). This property is leveraged by [GVW15] to construct partially hiding predicate (or attribute) encryption (PHPE), which allows computation of an inner product of a public attribute vector corresponding to $\mathsf{FHE.Eval}(C, \widehat{\mathbf{a}})$ and a private attribute vector, corresponding to the FHE secret key. Since inner product loosely approximates FHE decryption, this allows the decryptor to obtain a plaintext value corresponding to $C(\mathbf{a})$ as desired.

While the predicate encryption scheme [GVW15] can handle an unbounded number of 0-queries from the adversary, it runs into at least three difficulties when faced with a 1-query:

1. The proof of security in the PHPE scheme uses a trapdoor puncturing technique [ABB10] in the simulation, so that the simulator has a trapdoor to sample keys for 0-queries but this trapdoor vanishes for 1-queries, disabling the simulator.

2. Given a PHPE ciphertext $\mathsf{CT}_{\widehat{\mathbf{a}}}$ with public attributes $\widehat{\mathbf{a}}$, key homomorphism [BGG+14, GVW15] enables the evaluation of a circuit C on the PHPE ciphertext resulting in a PHPE ciphertext $\mathsf{CT}_{C(\widehat{\mathbf{a}})}$ with attributes $C(\widehat{\mathbf{a}})$. By construction, this resultant ciphertext is an LWE sample with an error term which is fixed and public *linear* combination of the error terms used to construct $\mathsf{CT}_{\widehat{\mathbf{a}}}$. This error is learned by the adversary upon decryption, which creates leakage that cannot be simulated. Indeed, this leakage, when sufficient, can completely break LWE security and allow the adversary to learn $\mathbf{a}$ in the clear (see Section 4 for details).

3. Recall that the FHE decryption operation is a lightweight operation conducted using PHPE with the FHE secret key as the private attribute vector. While FHE decryption is lightweight, it is still not lightweight enough to be performed in its entirety while maintaining the privacy of the FHE secret. FHE decryption is an inner product followed by a threshold function, of which only the inner product can be performed securely by PHPE. The authors overcome this hurdle by making use of the "lazy OR" trick, which roughly allows the decryptor to learn not the threshold inner product, but the pure inner product, which leaks sensitive information about the noise used while encrypting $\mathbf{a}$. Again, this leakage cannot be simulated, and when sufficiently high, can lead to complete recovery of the FHE secret key.

Attacks. Interestingly, all of the above difficulties in proving security translate to polynomial time attacks that lead to complete message recovery in a game where 1-keys are permitted. Our first and strongest attack is related to the first difficulty, and is effective even against the inner product predicate encryption scheme of Agrawal et al. [AFV11]. The attack exploits the fact that the real world key generation algorithm is stateless, and upon being asked for a key corresponding to the *same* function vector multiple times, provides independent short vectors in the same coset of a given lattice (say $\mathbf{F}$), which can be combined to recover a full trapdoor for $\mathbf{F}$[1]. Since the key in question is a 1-key, the construction allows the attacker to recover an LWE sample $\mathbf{F}^{\intercal}\mathbf{s} + \mathsf{noise}$ just by following the decryption procedure. This LWE sample unresistingly reveals all its secrets given a trapdoor for $\mathbf{F}$, which in turn allow the attacker to recover the entire message.

Our second attack is against the Partially Hiding Predicate Encryption system for circuits [GVW15] and stems from the second difficulty above. This attack exploits the fact that the decryptor, given a 1-key, learns a public linear function of the error terms used in encryption. By requesting sufficient 1-keys, the attacker can solve this linear system to recover the errors used in encryption, which lead to recovery of the predicate $\mathbf{a}$ even when functionality reveals much less.

Our third attack is against the Predicate Encryption system for circuits [GVW15]. As discussed in the third difficulty, the PE decryption key, which wishes to provide the decryptor with a threshold inner product value, instead can only provide the exact inner product value, leaving the decryptor to compute the threshold herself. This leads to an attacker learning linear equations in the errors used to construct the FHE encryption $\widehat{\mathbf{a}}$, which, when sufficiently many, let her recover the FHE secret, which in turn lets her recover $\mathbf{a}$.

We emphasize that our attacks are entirely practical. For instance consider the following scenario. A user who is authorized to obtain a key corresponding to circuit C, pretends that she has lost her key and re-issues a key request. The key generator checks her credentials and returns a new key SK_C. By obtaining multiple keys for the same circuit C as in Attack #1, the malicious user, who is only authorised to learn the single bit $C(\mathbf{a})$, instead learns the entire message $\mathbf{a}$. Our attacks also apply to the weaker indistinguishability based security game of functional encryption [BSW11] but do not work in the "weak attribute hiding" security game considered by [AFV11, GVW15].

[1]The attack can be extended to the setting where multiple, linearly dependent vectors are queried.

Selectively Secure $(1, \text{poly})$-Functional Encryption. We provide a construction that overcomes the above vulnerabilities for the case of a single 1-key, and continues to support an unbounded number of 0-keys. By restricting the attacker to any single query, this yields an alternate construction for reusable garbled circuits [GTKP+13]. We summarize the main ideas here. For clarity of exposition, we omit many details; we refer the reader to Sections 5 and 7 for the formal construction and proof.

Our starting point is the predicate encryption scheme of [GVW15], which we will hereby refer to as $(0, \text{poly})$-FE, as it supports zero 1-queries and any polynomial number of 0-queries. The construction for $(0, \text{poly})$-FE makes use of two components as described above, namely, $(0, \text{poly})$-partially hiding predicate encryption (PHPE) and fully homomorphic encryption (FHE). Our construction for $(1, \text{poly})$-FE follows the same high level template as [GVW15], and as our first step, we require $(1, \text{poly})$-PHPE. Note that the $(0, \text{poly})$-PHPE scheme does allow the key generator to release an unbounded number of both 0 and 1 queries, but as mentioned above, the proof of security breaks down if the adversary requests a 1-key. This is because the secret key corresponding to a circuit C is a low norm matrix $\mathbf{K}$ satisfying an equation of the following form:

$$\left[\mathbf{A} \mid \mathbf{A}_C \right] \mathbf{K} = \mathbf{P} \mod q$$

where the matrices $\mathbf{A}, \mathbf{P}$ are fixed and public, and the matrix $\mathbf{A}_C$ is computed by executing a homomorphic evaluation procedure [BGG+14, GVW15] corresponding to the circuit C on some public matrices. In the real system, the key generator has a trapdoor for $\mathbf{A}$, which allows it to sample $\mathbf{K}$ using known techniques [CHKP10, ABB10]. In the simulation, the matrix $\mathbf{A}_C$ has a special form, namely $\mathbf{A}_C = [\mathbf{A}\mathbf{R}_C - C(\mathbf{a}) \cdot \mathbf{G}]$ for some low norm matrix $\mathbf{R}_C$ and fixed public matrix $\mathbf{G}$. The simulator has a trapdoor for $\mathbf{G}$ which enables it to sample the required $\mathbf{K}$ also using known techniques *but only when* $C(\mathbf{a}) \neq 0$ [ABB10]. When $C(\mathbf{a}) = 0$, $\mathbf{G}$ vanishes along with its trapdoor, and the simulator has no method by which to sample $\mathbf{K}$[2].

To overcome this, we note that if the circuit C is known before the public key is generated, the simulator can instead sample $\mathbf{K}$ first and set $\mathbf{P}$ to satisfy the above equation. This is a standard trick in LWE based systems [GPV08, Pei13], and yields the same distribution of the pair $(\mathbf{K}, \mathbf{P})$ as in the real world. This allows us to take a step forward[3], but the adversary's view remains distinguishable from the real world, because decryption leaks correlated noise which is un-simulatable, as discussed in difficulty #2 above. To overcome this, we must choose the noise in the challenge ciphertext with care so that the noise yielded by the decryption equation is statistically close to fresh and independently chosen noise. Put together, these tricks enable us to build a $(1, \text{poly})$-PHPE.

However, $(1, \text{poly})$-PHPE does not immediately yield $(1, \text{poly})$-FE due to difficulty #3 above, namely, leakage on FHE noise. To handle this, we modify the circuit for which the PHPE key is provided so that the FHE ciphertext $\widehat{C(\mathbf{a})}$ is flooded with large noise before the inner product with the FHE secret key is computed. Now, though the attacker learns the exact noise in the evaluated FHE ciphertext $\widehat{C(\mathbf{a})}$ as before, this noise is independent of the noise used to generate $\widehat{\mathbf{a}}$ and no longer leaks any sensitive information. Note that care is required in executing the noise flooding step, since correctness demands that the FHE modulus be of polynomial size. To ensure this, we flood the FHE ciphertext *before* the FHE modulus reduction step. Now, we have at our disposal a $(1, \text{poly})$-FE scheme.

[2]The careful reader may observe that the simulator is disabled when $C(\mathbf{a}) = 0$, not when $C(\mathbf{a}) = 1$, though we have claimed that [AFV11, GVW15] can support 0-queries and not 1 queries. This is because, traditional functional encryption literature defines decryption to be permitted when the function value is 1, and defines the function value to be 1 when $C(\mathbf{a}) = 0$. We follow this flip to be consistent with prior work.

[3]This is presently a weak security game where the circuit C is announced before the parameters are generated. This restriction will be removed subsequently.

Upgrading Selective to Semi-Adaptive. The above techniques yield a $(1, \text{poly})$-FE scheme, but one that is secure according to a weak selective definition of security, which is inherited from the underlying $(1, \text{poly})$-PHPE scheme. We improve this by providing a method for compiling our selectively secure PHPE construction to one that satisfies *semi-adaptive* security, in which the attacker may see the public parameters before revealing the challenge. Our transformation is generic – it applies to all constructions that satisfy certain structural properties. In more detail, we require that: 1) the PHPE ciphertext $\mathsf{CT}_{\mathbf{a}}$ be decomposable into $|\mathbf{a}|$ components CT_i, where CT_i depends only on $\mathbf{a}[i]$, and 2) CT_i is a *fixed and public* linear function of the message $\mathbf{a}[i]$ and randomness chosen for encryption.

Concretely, consider the ciphertext in the $(0, \text{poly})$-PHPE of [GVW15]. For $i \in [\ell]$,

$$\mathsf{CT}_i = \mathbf{u}_i = \big(\mathbf{A}_i + \mathbf{a}[i] \cdot \mathbf{G}\big)^\mathsf{T}\mathbf{s} + \mathsf{noise}_i \quad \in \mathbb{Z}_q^m$$

Clearly condition (1) is satisfied – the i^{th} component of $\mathbf{a}$ influences only $\mathbf{u}_i$. Additionally, note that

$$\mathbf{u}_i = \big\langle [\ \mathbf{A}_i^\mathsf{T},\ 1,\ 1\ ];\ [\ \mathbf{s},\ \mathbf{a}[i] \cdot \mathbf{G}^\mathsf{T}\mathbf{s},\ \mathsf{noise}_i\ ] \big\rangle \mod q$$

Here, the first vector is a fixed public vector that is known to the key generator, while the second vector is made up of components all of which are known to the encryptor.

Given these two conditions, we construct a semi-adaptive PHPE for the circuit class $\mathcal{C}$, denoted by SaPH, using two ingredients:

1. A single key fully secure[4] functional encryption scheme, denoted by FuLin, for the inner product functionality defined as:

$$F_{(\mathbf{V}_1,\ldots,\mathbf{V}_k)}(\mathbf{a}_1, \ldots, \mathbf{a}_k) = \sum_{i \in [k]} \mathbf{V}_i \cdot \mathbf{a}_i \mod q$$

   Such a scheme was recently constructed by Agrawal et al. [ALS16].

2. A $(1, \text{poly})$ selectively secure PHPE scheme for the circuit class $\mathcal{C}$, which we denote by SelPH.

Intuitively, the idea is to nest the selective PHPE system for $\mathcal{C}$ within an adaptive FE system for inner products, so that the latter is used to generate ciphertexts of the former *on the fly*. In more detail, the public parameters of SaPH are set as the public parameters of FuLin, the secret key corresponding to C, namely $\mathsf{SaPH.SK}(C)$ is the tuple $\big(\ \mathsf{SelPH.PK},\ \mathsf{FuLin.SK}(\ [\ \mathbf{A}_i^\mathsf{T},\ 1,\ 1\ ]\ ),\ \mathsf{SelPH.SK}(C)\ \big)$ and the ciphertext is $\mathsf{SaPH.CT} = \mathsf{FuLin.CT}\big([\ \mathbf{s},\ \mathbf{a}[i] \cdot \mathbf{G}^\mathsf{T}\mathbf{s},\ \mathsf{noise}_i\ ]\big)$. Now, the ciphertext $\mathsf{FuLin.CT}\big([\ \mathbf{s},\ \mathbf{a}[i] \cdot \mathbf{G}^\mathsf{T}\mathbf{s},\ \mathsf{noise}_i\ ]\big)$ and secret key component $\mathsf{FuLin.SK}(\ [\ \mathbf{A}_i^\mathsf{T},\ 1,\ 1\ ]\ )$ may be decrypted to obtain the SelPH ciphertext, which may be decrypted using $\mathsf{SelPH.SK}(C)$. Some care is required in ascertaining that FuLin is only invoked for a single secret key, but this can be ensured by taking multiple copies of the FuLin scheme, and using the same randomness to generate multiple copies of the same key.

The advantage to the above strategy is that the public parameters of the SaPH scheme are now set as the public parameters of the FuLin scheme, and the public parameters of the SelPH scheme are moved into the secret keys of SaPH scheme. This enables the simulator of the SaPH scheme to provide the public parameters using the (adaptive/full) simulator for the FuLin scheme, and delay programming the PHPE public parameters until after the challenge is received, as required by the SelPH simulator. Thus, selective security may be upgraded to semi-adaptive security for the circuit

[4] Please see Appendix 8.4 for the definition of full security.

class $\mathcal{C}$, by leveraging adaptive security of the simpler inner product functionality. For more details, please see Section 6. We note that concurrently and independently to our work, [BV16, GKW16] also construct semi-adaptive FE using completely different techniques.

Generalising to Q queries. To construct (Q, poly)-FE, we again begin by constructing (Q, poly)-PHPE, which in turn is constructed from $(1, \text{poly})$-PHPE. The $(1, \text{poly})$-PHPE scheme has the following structure: it encodes the message $\mathbf{b}$ within an LWE sample $\boldsymbol{\beta}_0 = \mathbf{P}^\intercal \mathbf{s} + \mathsf{noise} + \mathbf{b}$. Given other components of the ciphertext, the decryptor is able to compute a ciphertext $\mathbf{c}_{\mathsf{Eval}}$ and key generator provides as the key a short matrix $\mathbf{K}$, where

$$\mathbf{c}_{\mathsf{Eval}} = [\ \mathbf{A} \mid \mathbf{A}_C\ ]^\intercal \mathbf{s} + \mathsf{noise}, \quad [\ \mathbf{A} \mid \mathbf{A}_C\ ]\ \mathbf{K} = \mathbf{P} \mod q$$

By computing $\mathbf{K}^\intercal \mathbf{c}_{\mathsf{Eval}} - \boldsymbol{\beta}_0$ and rounding the result, the decryptor recovers $\mathbf{b}$.

To generalise the above to handle Q queries, a natural approach would be to encode the message Q times, using Q distinct matrices $\mathbf{P}_1, \ldots, \mathbf{P}_Q$ and have the i^{th} key $\mathbf{K}_i$ be a short matrix satisfying $[\ \mathbf{A} \mid \mathbf{A}_{C_i}\ ]\ \mathbf{K}_i = \mathbf{P}_i \mod q$. Then, the key generator can pick $\mathbf{P}_i$ for the i^{th} key, and sample the corresponding $\mathbf{K}_i$ as the secret key. However, this straightforward idea would require the key generator to keep track of how many keys it has produced so far and would make the key generator stateful, which is undesirable.

To get around this, we make use of a trick suggested by Gorbunov et al. [GVW12] in a similar context. The idea is to enable the key generator to generate a fresh matrix $\mathbf{P}_i^*$ for the i^{th} key in a stateless manner, as follows. We publish a set of matrices $\{\mathbf{P}_1, \ldots, \mathbf{P}_k\}$ in the public key, for some parameter k. The key generator chooses a random subset $\Delta_i \subset [k]$ s.t. $|\Delta_i| = v$ for some suitably chosen v, and computes $\mathbf{P}_i^* = \sum_{j \in \Delta_i} \mathbf{P}_j$. It then samples $\mathbf{K}_i$ so that

$$[\ \mathbf{A} \mid \mathbf{A}_{C_i}\ ]\ \mathbf{K}_i = \mathbf{P}_i^* \mod q$$

If we choose (v, k) as functions of the security parameter κ and number of queries Q in a way that the Q subsets $\Delta_1, \ldots, \Delta_Q$ are cover free with high probability, then this ensures that the matrices $\mathbf{P}_1^*, \ldots, \mathbf{P}_Q^*$ are independent and uniformly distributed, which will enable the simulator to sample the requisite keys. This idea can be converted to a secure scheme, see Section 5.3 for details.

This gives us a (Q, poly)-PHPE but constructing (Q, poly)-FE requires some more work. Instead of flooding the evaluated ciphertext with a single piece of noise, we must now encode at least Q pieces of noise, to flood the ciphertext for Q decryptions. Fortunately, this can be ensured by leveraging cover-free sets again, so that the decryptor is forced to add a random cover-free subset sum of noise terms to the ciphertext before decryption. This ensures that each decryption lets the decryptor learn a fresh noise term which wipes out any troublesome noise leakage. Details are in Section 7.3.

Organization of the paper. The paper is organized as follows. In Section 2, we provide the priliminary definitions and lemmas related to lattices that will be required in the remainder of the paper. In Section 3 we provide definitions of $(1, \text{poly})$ partially hiding predicate encryption, $(1, \text{poly})$ functional encryption and fully homomorphic encryption as well as the algorithms that will be used in our paper. In Section 4, we describe our three attacks using 1-keys against existing predicate encryption systems. In Section 5 we provide our construction for $(1, \text{poly})$ partially hiding predicate encryption as well as its generalisation to (Q, poly)-PHPE. This is upgraded to achieve semi-adaptive security in Section 6. In Section 7 we provide our (Q, poly) FE scheme.

2 Lattice Preliminaries

Notation. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q and we represent $\mathbb{Z}_q$ as integers in $(-q/2, q/2]$. We let $\mathbb{Z}_q^{n \times m}$ denote the set of $n \times m$ matrices with entries in $\mathbb{Z}_q$. We use bold capital letters (e.g. $\mathbf{A}$) to denote matrices, bold lowercase letters (e.g. $\mathbf{x}$) to denote vectors. We use $\mathbf{x}[i]$ to refer to the i^{th} element of vector $\mathbf{x}$. The notation $\mathbf{A}^\intercal$ denotes the transpose of the matrix $\mathbf{A}$. When we say a matrix defined over $\mathbb{Z}_q$ has *full rank*, we mean that it has full rank modulo each prime factor of q.

If $\mathbf{A}_1$ is an $n \times m$ matrix and $\mathbf{A}_2$ is an $n \times m'$ matrix, then $[\mathbf{A}_1 \| \mathbf{A}_2]$ denotes the $n \times (m + m')$ matrix formed by concatenating $\mathbf{A}_1$ and $\mathbf{A}_2$. If $\mathbf{x}_1$ is a length m vector and $\mathbf{x}_2$ is a length m' vector, then we let $[\mathbf{x}_1 | \mathbf{x}_2]$ denote the length $(m + m')$ vector formed by concatenating $\mathbf{x}_1$ and $\mathbf{x}_2$. However, when doing matrix-vector multiplication we always view vectors as column vectors.

For a vector $\mathbf{x}$, we let $\|\mathbf{x}\|$ denote its ℓ_2 norm and $\|\mathbf{x}\|_\infty$ denote its infinity norm. For any matrix $\mathbf{R}$, we define $\|\mathbf{R}\|$ (resp. $\mathbf{R}\|_\infty$) as the ℓ_2 (resp. infinity) length of the longest column of $\mathbf{R}$.

We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we use $\mathrm{negl}(n)$ to denote a negligible function of n. We say $f(n)$ is *polynomial* if it is $O(n^c)$ for some $c > 0$, and we use $\mathrm{poly}(n)$ to denote a polynomial function of n. We say an event occurs with *overwhelming probability* if its probability is $1 - \mathrm{negl}(n)$. The function $\log x$ is the base 2 logarithm of x. The notation $\lfloor x \rceil$ denotes the nearest integer to x, rounding towards 0 for half-integers.

We will denote by $\{\mathbf{A}_i\}$ and $\{\mathbf{v}_j\}$ the set of matrices $\{\mathbf{A}_i\}_{i \in [\ell}$ and the set of vectors $\{\mathbf{v}_j\}_{j \in [t]}$ when the range is clear from context.

2.1 Lattice Definitions

An *m-dimensional lattice* Λ is a full-rank discrete subgroup of $\mathbb{R}^m$. A *basis* of Λ is a linearly independent set of vectors whose span is Λ. We will be concerned with *integer lattices*, i.e., those whose points have coordinates in $\mathbb{Z}^m$. Among these lattices are the "q-ary" lattices defined as follows: for any integer $q \geq 2$ and any $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we define

$$\begin{aligned} \Lambda_q^\perp(\mathbf{A}) &:= \left\{\mathbf{e} \in \mathbb{Z}^m : \mathbf{A} \cdot \mathbf{e} = \mathbf{0} \bmod q\right\} \\ \Lambda_q^{\mathbf{u}}(\mathbf{A}) &:= \left\{\mathbf{e} \in \mathbb{Z}^m : \mathbf{A} \cdot \mathbf{e} = \mathbf{u} \bmod q\right\}. \end{aligned}$$

The lattice $\Lambda_q^{\mathbf{u}}(\mathbf{A})$ is a coset of $\Lambda_q^\perp(\mathbf{A})$; namely, $\Lambda_q^{\mathbf{u}}(\mathbf{A}) = \Lambda_q^\perp(\mathbf{A}) + \mathbf{t}$ for any $\mathbf{t}$ such that $\mathbf{A} \cdot \mathbf{t} = \mathbf{u} \bmod q$. Let $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_k\}$ be a set of vectors in $\mathbb{R}^m$ and $\widetilde{\mathbf{T}} := \{\tilde{\mathbf{t}}_1, \ldots, \tilde{\mathbf{t}}_k\} \subset \mathbb{R}^m$ be the Gram-Schmidt orthogonalization of $\mathbf{T}$. We define $\|\mathbf{T}\|_{\mathsf{GS}} = \|\widetilde{\mathbf{T}}\|$.

2.2 The LWE Problem

The *Learning with Errors* problem, or LWE, is the problem of distinguishing noisy inner products from random. More formally, we define the (average-case) problem as follows:

Definition 2.1 ([Reg09])**.** Let $n \geq 1$ and $q \geq 2$ be integers, and let χ be a probability distribution on $\mathbb{Z}_q$. For $\mathbf{r} \in \mathbb{Z}_q^n$, let $A_{\mathbf{r},\chi}$ be the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing a vector $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in \mathbb{Z}_q$ according to χ, and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{r} \rangle + e)$.

The *decision*-$\mathsf{LWE}_{q,n,\chi}$ *problem* is: for uniformly random $\mathbf{r} \in \mathbb{Z}_q^n$, given a $\mathrm{poly}(n)$ number of samples that are either (all) from $A_{\mathbf{r},\chi}$ or (all) uniformly random in $\mathbb{Z}_q^n \times \mathbb{Z}_q$, output 0 if the former holds and 1 if the latter holds.

We say the decision-$\mathsf{LWE}_{q,n,\chi}$ problem is *infeasible* if for all polynomial-time algorithms $\mathcal{A}$, the probability that $\mathcal{A}$ solves the decision-LWE problem (over $\mathbf{r}$ and $\mathcal{A}$'s random coins) is negligibly close to $1/2$ as a function of n.

Connection to Lattices. The power of the LWE problem comes from the fact that for certain noise distributions χ, solving the search-LWE problem is as hard as finding approximate solutions to the shortest independent vectors problem (SIVP) and the decision version of the shortest vector problem (GapSVP) in the worst case.

Let $B = B(n) \in \mathbb{N}$. A family of distributions $\chi = \{\chi_n\}_{n\in\mathbb{N}}$ is called B-bounded if

$$\Pr[\ \chi \in \{-B, \ldots, B\}\ ] = 1$$

Regev and Peikert [Reg09, Pei09] showed that there is a B bounded distribution χ such that solving $\mathsf{LWE}_{q,n,\chi}$ is as hard as (quantumly) approximating certain worst case lattice problems to a factor of $\tilde{O}(n \cdot q/B)$. These lattice problems are believed to be hard to approximate even for subexponential q/B, i.e. 2^{n^ϵ} for some fixed $0 < \epsilon < 1/2$.

Gaussian distributions. Regev [Reg09] defined a natural distribution on any m dimensional lattice Λ, called the *discrete Gaussian distribution*, parametrized by a scalar $\sigma > 0$. [GVW15] consider a slightly modified version of the discrete Gaussian, namely the truncated discrete Gaussian distribution, denoted by $\mathcal{D}_{\Lambda,\sigma}$, where the output is replaced by $\mathbf{0}$ whenever the $\|\cdot\|_\infty$ norm of the sample exceeds $\sigma \cdot \sqrt{m}$. We will find it convenient to work with the truncated discrete Gaussian.

The following lemma about discrete Gaussians will be very useful to us.

Lemma 2.2 (Drowning Lemma)**.** *[GKPV10] Let $n \in \mathbb{N}$. For any real $\sigma = \omega(\sqrt{\log n})$, and any $\mathbf{c} \in \mathbb{Z}^n$,*

$$\mathsf{SD}(\mathcal{D}_{\mathbb{Z}^n,\sigma},\ \mathcal{D}_{\mathbb{Z}^n,\sigma,\mathbf{c}}) \leq \|\mathbf{c}\|/\sigma$$

We will also need the following lemma about randomness extraction.

Lemma 2.3. *[ABB10] Suppose that $m > (n+1)\log q + \omega(\log n)$ and that $q > 2$ is prime. Let $\mathbf{R}$ be an $m \times k$ matrix chosen uniformly in $\{1,-1\}^{m\times k} \mod q$ where $k = k(n)$ is polynomial in n. Let $\mathbf{A}$ and $\mathbf{B}$ be matrices chosen uniformly in $\mathbb{Z}_q^{n\times m}$ and $\mathbb{Z}_q^{n\times k}$ respectively. Then, for all vectors $\mathbf{e} \in \mathbb{Z}^m$, the distribution $(\mathbf{A}, \mathbf{AR}, \mathbf{R}^\intercal\mathbf{e})$ is statistically close to the distribution $(\mathbf{A}, \mathbf{B}, \mathbf{R}^\intercal\mathbf{e})$.*

3 Preliminaries: Definitions and Algorithms

In this section we define the notions of partially hiding predicate encryption, $(1, \mathrm{poly})$-functional encryption and fully homomorphic encryption. We also provide the algorithms that we will require in the remainder of the work.

3.1 Partially Hiding Predicate Encryption

A Partially-Hiding Predicate Encryption scheme PHPE for a pair of input-universes $\mathcal{X}, \mathcal{Y}$, a predicate universe $\mathcal{C}$, a message space $\mathcal{M}$, consists of four algorithms (PH.Setup, PH.Enc, PH.KeyGen, PH.Dec):

$\mathsf{PH.Setup}(1^\kappa, \mathcal{X}, \mathcal{Y}, \mathcal{C}, \mathcal{M}) \to (\mathsf{PH.PK}, \mathsf{PH.MSK})$. The setup algorithm gets as input the security parameter κ and a description of $(\mathcal{X}, \mathcal{Y}, \mathcal{C}, \mathcal{M})$ and outputs the public parameter PH.PK, and the master key PH.MSK.

$\mathsf{PH.Enc}(\mathsf{PH.PK}, (\mathbf{x}, \mathbf{y}), \mu) \to \mathsf{CT}_{\mathbf{y}}$. The encryption algorithm gets as input $\mathsf{PH.PK}$, an attribute pair $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$ and a message $\mu \in \mathcal{M}$. It outputs a ciphertext $\mathsf{CT}_{\mathbf{y}}$.

$\mathsf{PH.KeyGen}(\mathsf{PH.MSK}, C) \to \mathsf{SK}_C$. The key generation algorithm gets as input $\mathsf{PH.MSK}$ and a predicate $C \in \mathcal{C}$. It outputs a secret key SK_C.

$\mathsf{PH.Dec}((\mathsf{SK}_C, C), (\mathsf{CT}, \mathbf{y})) \to \mu \vee \perp$. The decryption algorithm gets as input the secret key SK_C, a predicate C, and a ciphertext $\mathsf{CT}_{\mathbf{y}}$ and the public part $\mathbf{y}$ of the attribute vector. It outputs a message $\mu \in \mathcal{M}$ or $\perp$.

Correctness. We require that for all $(\mathsf{PH.PK}, \mathsf{PH.MSK}) \leftarrow \mathsf{PH.Setup}(1^\kappa, \mathcal{X}, \mathcal{Y}, \mathcal{C}, \mathcal{M})$, for all $(\mathbf{x}, \mathbf{y}, C) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{C}$ and for all $\mu \in \mathcal{M}$,

- For 1-queries, namely $C(\mathbf{x}, \mathbf{y}) = 1$,

$$\Big[\mathsf{PH.Dec}\big((\mathsf{SK}_C, C), (\mathsf{CT}_{\mathbf{y}}, \mathbf{y})\big) = \mu\Big] \geq 1 - \mathrm{negl}(\kappa)$$

- For 0-queries, namely $C(\mathbf{x}, \mathbf{y}) = 0$,

$$\Big[\mathsf{PH.Dec}\big((\mathsf{SK}_C, C), (\mathsf{CT}_{\mathbf{y}}, \mathbf{y})\big) = \perp\Big] \geq 1 - \mathrm{negl}(\kappa)$$

Semi-Adaptive SIM Security. Below, we define the semi-adaptive experiment for partially hiding predicate encryption (PHPE) that supports a single 1-query and an unbounded number of 0-queries. We denote such a scheme by $(1, \mathrm{poly})$-PHPE scheme. We note that the scheme of Gorbunov et al. [GVW15] is a $(0, \mathrm{poly})$-PHPE scheme.

Definition 3.1 ($(1, \mathrm{poly})$ (Semi-Adaptive)-SIM-Attribute-Hiding)**.** Let PHPE be a partially hiding predicate encryption scheme for a circuit family $\mathcal{C}$. For every p.p.t. adversary Adv and a stateful p.p.t. simulator Sim, consider the following two experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{PHPE},\mathsf{Adv}}(1^\kappa)}$**:**

1: $(\mathsf{PH.PK}, \mathsf{PH.MSK}) \leftarrow \mathsf{PH.Setup}(1^\kappa)$
2: $(\mathbf{x}, \mathbf{y}, \mu, C^*, \mathsf{st}_{\mathsf{A}}) \leftarrow \mathsf{Adv}(\mathsf{PH.PK})$
3: $\mathsf{CT}_{\mathbf{y}} \leftarrow \mathsf{PH.Enc}\big(\mathsf{PH.PK}, (\mathbf{x}, \mathbf{y}), \mu\big)$
4: $\mathsf{SK}_{C^*} \leftarrow \mathsf{PH.KeyGen}\big(\mathsf{PH.MSK}, C^*\big)$
5: $\alpha \leftarrow \mathsf{Adv}^{\mathsf{PH.KeyGen}(\mathsf{PH.MSK},\cdot)}(\mathsf{CT}_{\mathbf{y}}, \mathsf{SK}_{C^*}, \mathsf{st}_{\mathsf{A}})$
6: Output $(\mathbf{x}, \mathbf{y}, \mu, \alpha)$

$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{PHPE},\mathsf{Sim}}(1^\kappa)}$**:**

1: $\mathsf{PH.PK} \leftarrow \mathsf{Sim}(1^\kappa)$
2: $(\mathbf{x}, \mathbf{y}, \mu, C^*, \mathsf{st}_{\mathsf{A}}) \leftarrow \mathsf{Adv}(\mathsf{PH.PK})$
3: $\mathsf{CT}_{\mathbf{y}} \leftarrow \mathsf{Sim}\ (\mathbf{y}, 1^{|\mathbf{x}|}, C^*, \mu)$
4: $\mathsf{SK}_{C^*} \leftarrow \mathsf{Sim}$
5: $\alpha \leftarrow \mathsf{Adv}^{\mathsf{Sim}}(\mathsf{CT}_{\mathbf{y}}, \mathsf{SK}_{C^*}, \mathsf{st}_{\mathsf{A}})$
6: Output $(\mathbf{x}, \mathbf{y}, \mu, \alpha)$

We say an adversary Adv is admissible if:

1. For the single query C^*, it holds that $C^*(\mathbf{x}, \mathbf{y}) = 1$.
2. For all queries $C \neq C^*$, it holds that $C(\mathbf{x}, \mathbf{y}) = 0$.

In the ideal experiment, the simulator Sim is traditionally given access to an oracle $U_{(\mathbf{x},\mathbf{y},\mu)}(\cdot)$, which upon input C returns $\perp$ if $C(\mathbf{x},\mathbf{y}) = 0$ and μ if $C(\mathbf{x},\mathbf{y}) = 1$. However, since in our case Sim is provided C^* explicitly, and this is the only 1-query, the simulator can check whether $C_i = C^*$ for any query C_i, and if not, set $C_i(\mathbf{x},\mathbf{y}) = 0$. Hence, to simplify notation, we omit the oracle in the ideal experiment above. Note that since C^* is a 1-query by definition, the simulator must be provided μ.

The partially hiding predicate encryption scheme PHPE is said to be $(1,\mathrm{poly})$-attribute hiding if there exists a p.p.t. simulator Sim such that for every admissible p.p.t. adversary Adv, the following two distributions are computationally indistinguishable:

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{PHPE},\mathsf{Adv}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{PHPE},\mathsf{Sim}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$

SIM Security (Selective). Next, we define a selective variant of the above game.

Definition 3.2 ($(1,\mathrm{poly})$ (Selective)-SIM-Attribute-Hiding)**.** Let PHPE be a partially hiding predicate encryption scheme for a circuit family $\mathcal{C}$. For every p.p.t. adversary Adv and a stateful p.p.t. simulator Sim, consider the following two experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{PHPE},\mathsf{Adv}}(1^\kappa)\textbf{:}}$

1: $(\mathbf{x},\mathbf{y},\mu,C^*,\mathsf{st}_\mathsf{A}) \leftarrow \mathsf{Adv}(1^\kappa)$
2: $(\mathsf{PH.PK},\mathsf{PH.MSK}) \leftarrow \mathsf{PH.Setup}(1^\kappa)$
3: $\mathsf{CT}_\mathbf{y} \leftarrow \mathsf{PH.Enc}\big(\mathsf{PH.PK},(\mathbf{x},\mathbf{y}),\mu\big)$
4: $\mathsf{SK}_{C^*} \leftarrow \mathsf{PH.KeyGen}\big(\mathsf{PH.MSK},C^*\big)$
5: $\alpha \leftarrow \mathsf{Adv}^{\mathsf{PH.KeyGen}(\mathsf{PH.MSK},\cdot)}(\mathsf{CT}_\mathbf{y},\mathsf{SK}_{C^*},\mathsf{st}_\mathsf{A})$
6: Output $(\mathbf{x},\mathbf{y},\mu,\alpha)$

$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{PHPE},\mathsf{Sim}}(1^\kappa)\textbf{:}}$

1: $(\mathbf{x},\mathbf{y},\mu,C^*,\mathsf{st}_\mathsf{A}) \leftarrow \mathsf{Adv}(1^\kappa)$
2: $\mathsf{PH.PK} \leftarrow \mathsf{Sim}\big(1^\kappa,\mathbf{y},1^{|\mathbf{x}|},C^*,\mu\big)$
3: $\mathsf{CT}_\mathbf{y} \leftarrow \mathsf{Sim}$
4: $\mathsf{SK}_{C^*} \leftarrow \mathsf{Sim}$
5: $\alpha \leftarrow \mathsf{Adv}^{\mathsf{Sim}}(\mathsf{CT}_\mathbf{y},\mathsf{SK}_{C^*},\mathsf{st}_\mathsf{A})$
6: Output $(\mathbf{x},\mathbf{y},\mu,\alpha)$

The admissibility of the adversary Adv, the notes about the simulator and the required indistinguishability of distributions are as in Definition 3.1.

Note. We note that our game is more restricted than the standard selective game, in that the attacker must output not just $(\mathbf{x},\mathbf{y})$ but also C^* in the first step. In the traditional selective game, the query C^* may be specified after the adversary sees the public parameters. However, we choose not give our variant a new name for simplicity, since we will anyway achieve the semi-adaptive definition, which is stronger than the standard selective definition.

For the definition of (Q,poly)-PHPE, where an adversary may request Q decrypting queries, we merely replace each occurence of C^* with a tuple $C^*_1,\ldots,C^*_Q$ in both the games above.

3.2 $(1,\mathrm{poly})$-Functional Encryption

In this section, we define $(1,\mathrm{poly})$ functional encryption.

Definition 3.3. A functional encryption scheme FE for an input universe $\mathcal{X}$, a circuit universe $\mathcal{C}$ and a message space $\mathcal{M}$, consists of four algorithms $\mathsf{FE} = (\mathsf{FE.Setup},\mathsf{FE.Keygen},\mathsf{FE.Enc},\mathsf{FE.Dec})$ defined as follows.

- $\mathsf{FE.Setup}(1^\kappa)$ is a p.p.t. algorithm takes as input the unary representation of the security parameter and outputs the master public and secret keys $(\mathsf{PK}, \mathsf{MSK})$.
- $\mathsf{FE.Keygen}(\mathsf{MSK}, C)$ is a p.p.t. algorithm that takes as input the master secret key MSK and a circuit $C \in \mathcal{C}$ and outputs a corresponding secret key SK_C.
- $\mathsf{FE.Enc}\big(\mathsf{PK}, (\mathbf{a}, \mu)\big)$ is a p.p.t. algorithm that takes as input the master public key PK and an input message $(\mathbf{a}, \mu) \in \mathcal{X} \times \mathcal{M}$ and outputs a ciphertext $\mathsf{CT}_\mathbf{a}$.
- $\mathsf{FE.Dec}(\mathsf{SK}_C, \mathsf{CT}_\mathbf{a})$ is a deterministic algorithm that takes as input the secret key SK_C and a ciphertext $\mathsf{CT}_\mathbf{a}$ and outputs μ iff $C(\mathbf{a}) = 1$, $\perp$ otherwise.

Note that our definition is a slightly modified, albeit equivalent version of the standard definition for FE [BSW11]. For compatibility with the PHPE definition, we define our functionality to incorporate a message bit μ which is revealed when $C(\mathbf{a}) = 1$.

Correctness. Next, we define correctness of the system.

Definition 3.4 (Correctness)**.** A functional encryption scheme FE is correct if for all $C \in \mathcal{C}_\kappa$ and all $\mathbf{a} \in \mathcal{X}_\kappa$,

- If $C(\mathbf{a}) = 1$

$$\Pr\left[\begin{array}{l}(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\kappa);\\ \mathsf{FE.Dec}\Big(\mathsf{FE.Keygen}(\mathsf{MSK}, C), \mathsf{FE.Enc}\big(\mathsf{PK}, (\mathbf{a}, \mu)\big)\Big) \neq \mu\end{array}\right] = \mathrm{negl}(\kappa)$$

- If $C(\mathbf{a}) = 0$

$$\Pr\left[\begin{array}{l}(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\kappa);\\ \mathsf{FE.Dec}\Big(\mathsf{FE.Keygen}(\mathsf{MSK}, C), \mathsf{FE.Enc}\big(\mathsf{PK}, (\mathbf{a}, \mu)\big)\Big) \neq \perp\end{array}\right] = \mathrm{negl}(\kappa)$$

where the probability is taken over the coins of $\mathsf{FE.Setup}$, $\mathsf{FE.Keygen}$, and $\mathsf{FE.Enc}$.

3.2.1 Semi-Adaptive Security.

We define the notion of semi-adaptive simulation based security for $(1, \mathrm{poly})$-functional encryption, analogous to Definition 3.1.

Definition 3.5 (Semi-Adaptive SIM)**.** Let FE be a functional encryption scheme for a circuit family $\mathcal{C}$. For every p.p.t. adversary Adv and a stateful p.p.t. simulator Sim, consider the following two experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{Adv}}(1^\kappa)\textbf{:}}$

1: $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\kappa)$
2: $(\mathbf{a}, \mu, C^*, \mathsf{st}) \leftarrow \mathsf{Adv}(1^\kappa, \mathsf{PK})$
3: $\mathsf{CT}_\mathbf{a} \leftarrow \mathsf{FE.Enc}\big(\ \mathsf{PK}, \mathbf{a}, \mu\ \big)$
4: $\mathsf{SK}_{C^*} \leftarrow \mathsf{FE.Keygen}(\mathsf{MSK}, C^*)$
5: $\alpha \leftarrow \mathsf{Adv}^{\mathsf{FE.Keygen}(\mathsf{MSK},\cdot)}(\mathsf{CT}_\mathbf{a}, \mathsf{SK}_{C^*}, \mathsf{st})$
6: Output $(\mathbf{a}, \mu, \alpha)$

$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}(1^\kappa)\textbf{:}}$

1: $\mathsf{PK} \leftarrow \mathsf{Sim}(1^\kappa)$
2: $(\mathbf{a}, \mu, C^*, \mathsf{st}) \leftarrow \mathsf{Adv}(1^\kappa, \mathsf{PK})$
3: $\mathsf{CT}_\mathbf{a} \leftarrow \mathsf{Sim}(1^{|\mathbf{a}|}, C^*, \mu)$
4: $\mathsf{SK}_{C^*} \leftarrow \mathsf{Sim}$
5: $\alpha \leftarrow \mathsf{Adv}^{\mathsf{Sim}}(\mathsf{CT}_\mathbf{a}, \mathsf{SK}_{C^*}, \mathsf{st})$
6: Output $(\mathbf{a}, \mu, \alpha)$

We say an adversary Adv is admissible if:

1. For a single query, C^*, it holds that $C^*(\mathbf{a}) = 1$.
2. For all other queries $C_i \neq C^*$, it holds that $C_i(\mathbf{a}) = 0$.

In the ideal experiment, the simulator Sim is traditionally given access to an oracle $U_{(\mathbf{a},\mu)}(\cdot)$, which upon input C returns $\perp$ if $C(\mathbf{a}) = 0$ and μ if $C(\mathbf{a}) = 1$. However, as in Definition 3.1, we note that our simulator does not require access to an oracle because an admissible adversary may only make a single 1 query, denoted by C^*, which is provided explicitly to the simulator. Every other query C_i made by the adversary is a 0 query, hence the simulator can compare each query C_i with C^*, and set $C_i(\mathbf{a}) = 0$ when the equality does not hold.

The $(1, \text{poly})$-functional encryption scheme FE is then said to be $\mathsf{SA\text{-}SIM}$-secure if there is an admissible stateful p.p.t. simulator Sim such that for every admissible p.p.t. adversary Adv, the following two distributions are computationally indistinguishable.

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{Adv}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$

As before, for the (Q, poly) version of the above game, we merely replace each occurence of C^* with a tuple $C^*_1, \ldots, C^*_Q$.

3.3 Fully Homomorphic Encryption

A leveled symmetric key fully homomorphic scheme is a tuple of P.P.T algorithms $\mathsf{FHE.KeyGen}$, $\mathsf{FHE.Enc}$, $\mathsf{FHE.Eval}$ and $\mathsf{FHE.Dec}$:

$\mathsf{FHE.KeyGen}(1^\kappa, 1^d, 1^k)$: This is a probabilistic algorithm that takes as input the security parameter, the depth bound for the circuit, the message length and outputs the secret key $\mathsf{FHE.SK}$.

$\mathsf{FHE.Enc}(\mathsf{FHE.SK}, \mu)$: This is a probabilistic algorithm that takes as input the secret key and message and produces the ciphertext $\mathsf{FHE.CT}$.

$\mathsf{FHE.Eval}(C, \mathsf{FHE.CT})$: This is a deterministic algorithm that takes as input a Boolean circuit $C : \{0,1\}^k \to \{0,1\}$ of depth at most d and outputs another ciphertext $\mathsf{FHE.CT}'$.

$\mathsf{FHE.Dec}(\mathsf{FHE.SK}, \mathsf{FHE.CT})$: This is a deterministic algorithm that takes as input the secret key and a ciphertext and produces a bit.

Correctness. Let $\mathsf{FHE.SK} \leftarrow \mathsf{FHE.KeyGen}(1^\kappa, 1^d, 1^k)$ and C be a circuit of depth d. Then we require that

$$\Pr\Big[\ \mathsf{FHE.Dec}\big(\mathsf{FHE.SK},\ \mathsf{FHE.Eval}(C, \mathsf{FHE.Enc}(\mathsf{FHE.SK}, \mu))\big) = C(\mu)\Big] = 1$$

Security. Security is defined as the standard semantic security. Let $\mathcal{A}$ be an efficient, stateful adversary and $d, k = \text{poly}(\kappa)$. The semantic security game is defined as follows.

1. $\mathsf{FHE.SK} \leftarrow \mathsf{FHE.Setup}(1^\kappa, 1^d, 1^k)$
2. $(\mu_0, \mu_1) \leftarrow \mathcal{A}(1^\kappa, 1^d, 1^k)$

3. $b \leftarrow \{0,1\}$
4. $\mathsf{FHE.CT} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.SK}, \mu_b)$
5. $b' \leftarrow \mathcal{A}(\mathsf{FHE.CT})$

We require that the advantage of $\mathcal{A}$ in the above game be negligible, namely

$$|\Pr(b' = b) - 1/2| = \mathrm{negl}(\kappa)$$

Instantiating FHE from Learning with Errors. We make use of the following instantiation of FHE from LWE.

Theorem 3.6. *[BV11, BGV12, GSW13, BV14, AP14] There is an FHE scheme based on the LWE assumption such that, as long as $q \geq O(\kappa^2)$:*

1. $\mathsf{FHE.SK} \in \mathbb{Z}_q^t$ *for some* $t \in \mathrm{poly}(\kappa)$.
2. $\mathsf{FHE.CT}(\mu) \in \{0,1\}^\ell$ *where* $\ell = \mathrm{poly}(\kappa, k, d, \log q)$.
3. $\mathsf{FHE.Eval}$ *outputs a ciphertext* $\mathsf{FHE.CT}' \in \{0,1\}^\ell$.
4. *There exists an algorithm* $\mathsf{FHE.Scale}(q,p)$ *which reduces the modulus of the FHE ciphertext from* q *to* p.
5. *For any Boolean circuit of depth* d, $\mathsf{FHE.Eval}(C,\cdot)$ *is computed by a Boolean circuit of depth* $\mathrm{poly}(d, \kappa, \log q)$.
6. $\mathsf{FHE.Dec}$ *on input* $\mathsf{FHE.SK}$ *and* $\mathsf{FHE.CT}'$ *outputs a bit* $b \in \{0,1\}$. *If* $\mathsf{FHE.CT}'$ *is an encryption of* 1, *then*
$$\sum_{i \in [t]} \mathsf{FHE.SK}[i] \cdot \mathsf{FHE.CT}'[i] \in [\lfloor p/2 \rfloor - B, \lfloor p/2 \rfloor + B]$$
for some fixed $B = \mathrm{poly}(\kappa)$. *If* $\mathsf{FHE.CT}'$ *is an encryption of* 0, *then*
$$\sum_{i \in [t]} \mathsf{FHE.SK}[i] \cdot \mathsf{FHE.CT}'[i] \notin [\lfloor p/2 \rfloor - B, \lfloor p/2 \rfloor + B]$$
7. *Security relies on* $\mathsf{LWE}_{\Theta(t),q,\chi}$.

3.4 Algorithms used by our constructions

The following algorithms will be used crucially in our construction and proof.

3.4.1 Trapdoor Generation

Below, we discuss two kinds of trapdoors that our construction and proof will use.

Generating random lattices with trapdoors. To begin, we provide an algorithm for generating a random lattice with a trapdoor.

Theorem 3.7. *[Ajt99, GPV08, MP12] Let q, n, m be positive integers with $q \geq 2$ and $m \geq 6n \lg q$. There is a probabilistic polynomial-time algorithm $\mathsf{TrapGen}(q, n, m)$ that with overwhelming probability (in n) outputs a pair $(\mathbf{A} \in \mathbb{Z}_q^{n \times m},\ \mathbf{T} \in \mathbb{Z}^{m \times m})$ such that $\mathbf{A}$ is statistically close to uniform in $\mathbb{Z}_q^{n \times m}$ and $\mathbf{T}$ is a basis for $\Lambda_q^{\perp}((\mathbf{A}))$ satisfying*

$$\|\mathbf{T}\|_{\mathsf{GS}} \leq O(\sqrt{n \log q}) \quad \textit{and} \quad \|\mathbf{T}\| \leq O(n \log q).$$

The primitive matrix G and its trapdoor. The matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ is the "powers-of two" matrix (see [MP12, Pei13] for the definition). The matrix $\mathbf{G}$ has a public trapdoor $\mathbf{T_G}$ such that $\|\mathbf{T_G}\|_\infty = 2$. Let $\mathbf{G}^{-1} : \mathbb{Z}_q^{n \times m} \to \mathbb{Z}_q^{n \times m}$ denote a *deterministic* algorithm which outputs a short preimage $\tilde{\mathbf{A}}$ so that $\mathbf{G} \cdot \tilde{\mathbf{A}} = \mathbf{A} \mod q$.

3.4.2 Three ways of generating a distribution.

Let $\mathbf{F} = [\ \mathbf{A} | \mathbf{AR} + \gamma \cdot \mathbf{G}\ ]$ where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\mathbf{R} \leftarrow \{-1, 1\}^{m \times m}$, $\mathbf{G}$ is the primitive matrix defined above and $\gamma \in \mathbb{Z}_q$ is arbitrary (in particular, it can be 0). We are interested in the distribution $(\mathbf{F}, \mathbf{K}, \mathbf{P}) \in \mathbb{Z}_q^{n \times 2m} \times \mathbb{Z}_q^{2m \times m} \times \mathbb{Z}_q^{n \times m}$ satisfying $\mathbf{F}\ \mathbf{K} = \mathbf{P} \mod q$.

Given $\mathbf{F}$, we provide three different ways of sampling $(\mathbf{K}, \mathbf{P})$ so that the same resultant distribution is obtained.

1. The first method is to sample $\mathbf{P} \leftarrow \mathbb{Z}_q^{n \times m}$ randomly and use a trapdoor for the left matrix of $\mathbf{F}$, namely $\mathbf{A}$ to sample a low norm $\mathbf{K}$. We let $\mathbf{B} \triangleq \mathbf{AR} + \gamma \cdot \mathbf{G}$ and $\mathbf{p}$ denote a column of $\mathbf{P}$.

   Algorithm $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{p}, \sigma)$ [CHKP10, ABB10]:

   Inputs: a full rank matrix $\mathbf{A}$ in $\mathbb{Z}_q^{n \times m}$, a "short" basis $\mathbf{T_A}$ of $\Lambda_q^{\perp}(\mathbf{A})$, a matrix $\mathbf{B}$ in $\mathbb{Z}_q^{n \times m}$, a vector $\mathbf{p} \in \mathbb{Z}_q^n$, and a Gaussian parameter σ. (3.1)

   Output: The algorithm outputs a vector $\mathbf{k} \in \mathbb{Z}^{2m}$ in coset $\Lambda_q^{\mathbf{p}}(\mathbf{F})$.

   Its distribution is analyzed in the following theorem.

   Theorem 3.8 ([ABB10, Theorem 17], [CHKP10, Lemma 3.2])**.** *Let $q > 2$, $m > n$ and $\sigma > \|\mathbf{T_A}\|_{\mathsf{GS}} \cdot \omega(\sqrt{\log(2m)})$. Then $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{p}, \sigma)$ taking inputs as in (3.1), outputs a vector $\mathbf{k} \in \mathbb{Z}^{2m}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^{\mathbf{p}}(\mathbf{F}), \sigma}$ where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$.*

2. The second method is to again sample $\mathbf{P} \leftarrow \mathbb{Z}_q^{n \times m}$ and use a trapdoor for the right matrix $\mathbf{G}$ (when $\gamma \neq 0$) to sample $\mathbf{K}$.

   Algorithm $\mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}, \mathbf{T_G}, \mathbf{p}, \sigma)$**:**

   Inputs: matrices $\mathbf{A}$ in $\mathbb{Z}_q^{n \times k}$ and $\mathbf{R}$ in $\mathbb{Z}^{k \times m}$, a full rank matrix $\mathbf{G}$ in $\mathbb{Z}_q^{n \times m}$, a "short" basis $\mathbf{T_G}$ of $\Lambda_q^{\perp}(\mathbf{G})$, a vector $\mathbf{p} \in \mathbb{Z}_q^n$, and a Gaussian parameter σ. (3.2)

   Output: The algorithm outputs a vector $\mathbf{k} \in \mathbb{Z}^{2m}$ in coset $\Lambda_q^{\mathbf{p}}(\mathbf{F})$.

Often the matrix $\mathbf{R}$ given to the algorithm as input will be a random matrix in $\{1,-1\}^{m\times m}$. Let S^m be the m-sphere $\{\mathbf{x}\in\mathbb{R}^{m+1} \ : \ \|\mathbf{x}\|=1\}$. We define $s_R := \|\mathbf{R}\| = \sup_{\mathbf{x}\in S^{m-1}}\|\mathbf{R}\cdot\mathbf{x}\|$.

Theorem 3.9 ([ABB10, Theorem 19]). *Let $q>2, m>n$ and $\sigma > \|\mathbf{T_G}\|_{\mathsf{GS}}\cdot s_R\cdot\omega(\sqrt{\log m})$. Then $\mathsf{SampleRight}(\mathbf{A},\mathbf{G},\mathbf{R},\mathbf{T_G},\mathbf{p},\sigma)$ taking inputs as in (3.2) outputs a vector $\mathbf{k}\in\mathbb{Z}^{2m}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^{\mathbf{p}}(\mathbf{F}),\sigma}$ where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{AR}+\gamma\cdot\mathbf{G})$.*

3. The final method is to sample $\mathbf{K}\leftarrow\left(\mathcal{D}_{\mathbb{Z}^{2m},\sigma}\right)^m$ and set $\mathbf{P}=\mathbf{F}\cdot\mathbf{K} \mod q$. We note that this method works even if $\gamma=0$. As argued by [GPV08, Lemma 5.2], this produces the correct distribution.

Lemma 3.10. *Assume the columns of $\mathbf{F}$ generate $\mathbb{Z}_q^n$ and let $\sigma\geq\omega(\sqrt{n\log q})$. Then, for $\mathbf{k}\leftarrow\mathcal{D}_{\mathbb{Z}^{2m},\sigma}$, the distribution of the vector $\mathbf{p}=\mathbf{F}\cdot\mathbf{k} \mod q$ is statistically close to uniform over $\mathbb{Z}_q^n$. Furthermore, fix $\mathbf{p}\in\mathbb{Z}_q^n$ and let $\mathbf{t}$ be an arbitrary solution s.t. $\mathbf{F}\cdot\mathbf{t}=\mathbf{p} \mod q$. Then, the conditional distribution of $\mathbf{k}\leftarrow\mathcal{D}_{\mathbb{Z}^{2m},\sigma}$ given $\mathbf{F}\cdot\mathbf{k}=\mathbf{p} \mod q$ is $\mathbf{t}+\mathcal{D}_{\Lambda^\perp(\mathbf{F}),\sigma,-\mathbf{t}}$, which is precisely $\mathcal{D}_{\Lambda_q^{\mathbf{p}}(\mathbf{F}),\sigma}$.*

3.4.3 Public Key and Ciphertext Evaluation Algorithms

Our construction will make use of the public key and ciphertext evaluation algorithms from [BGG+14, GVW15]. Since these algorithms can be used as black boxes for our purposes, we only state their input/output behavior and properties. These algorithms were constructed by Boneh et al [BGG+14] in the context of attribute based encryption, and extended by Gorbunov et al. [GVW15] to the setting of partially hiding predicate encryption. In this setting, the attributes are divided into a private component $\mathbf{x}$ and a public component $\mathbf{y}$, and the functionality supports computation of a lightweight inner product composed with a heavy circuit $\widehat{C}$. Formally, [GVW15] construct algorithms $\mathsf{PHPE.Eval}_{\mathsf{PK}}$ and $\mathsf{PHPE.Eval}_{\mathsf{CT}}$ to support the following circuit family:

$$\widehat{C}\circ\mathsf{IP}(\mathbf{x},\mathbf{y}) = \langle\mathbf{x},\widehat{C}(\mathbf{y})\rangle.$$

They make crucial use of the fact that $\mathsf{PHPE.Eval}_{\mathsf{CT}}$ does not need $\mathbf{x}$ for its execution since the computation involving $\mathbf{x}$ is an inner product. To compute the inner product, the multiplication may be carried out keeping $\mathbf{x}$ private and letting $\widehat{C}(\mathbf{y})$ be public, and addition may be carried out keeping both attributes private. Additionally, the circuit $\widehat{C}$ operates entirely on public attributes $\mathbf{y}$.

In more detail, [GVW15, Sec 3.2] demonstrate the existence of the following efficient algorithms:

1. $\mathsf{Eval}_{\mathsf{PK}}$ takes as input $\ell+t$ matrices $\{\mathbf{A}_i\}$, $\{\mathbf{B}_j\}\in\mathbb{Z}_q^{n\times m}$ and a circuit $\widehat{C}\circ\mathsf{IP}\in\mathcal{C}$ and outputs a matrix $\mathbf{A}_{\widehat{C}\circ\mathsf{IP}}\in\mathbb{Z}_q^{n\times m}$.
2. $\mathsf{Eval}_{\mathsf{CT}}$ takes as input $\ell+t$ matrices $\{\mathbf{A}_i,\}\{\mathbf{B}_j\}\in\mathbb{Z}_q^{n\times m}$, $\ell+t$ vectors $\{\mathbf{u}_i\},\{\mathbf{v}_j\}$, the public attribute $\mathbf{y}\in\{0,1\}^\ell$ and a circuit $\widehat{C}\circ\mathsf{IP}\in\mathcal{C}$, and outputs a vector $\mathbf{u}_{\widehat{C}\circ\mathsf{IP}}\in\mathbb{Z}_q^m$.
3. $\mathsf{Eval}_{\mathsf{R}}$ takes as input $\ell+t$ matrices $\{\mathbf{R}_i\},\{\mathbf{R}'_j\}\in\mathbb{Z}_q^{m\times m}$, the matrix $\mathbf{A}$, the public attribute vector $\mathbf{y}\in\{0,1\}^\ell$ and a circuit $\widehat{C}\circ\mathsf{IP}\in\mathcal{C}$ and outputs a matrix $\mathbf{R}_{\widehat{C}\circ\mathsf{IP}}\in\mathbb{Z}_q^{m\times m}$.

such that the following properties hold:

$$\mathbf{u}_{\widehat{C}\circ\mathsf{IP}} = \left(\ \mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \widehat{C}\circ\mathsf{IP}(\mathbf{x},\mathbf{y})\cdot\mathbf{G}\ \right)^{\mathsf{T}}\mathbf{s}+\mathbf{e}_{\mathsf{Eval}} \tag{3.3}$$

When

$$\mathbf{A}_i = \mathbf{A} \cdot \mathbf{R}_i - \mathbf{y}[i] \cdot \mathbf{G}$$
$$\mathbf{B}_i = \mathbf{A} \cdot \mathbf{R}'_i - \mathbf{x}[i] \cdot \mathbf{G}$$
$$\text{Then } \mathbf{A}_{\widehat{C} \circ \mathsf{IP}} = \mathbf{A}\mathbf{R}_{\widehat{C} \circ \mathsf{IP}} - \widehat{C} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) \cdot \mathbf{G} \tag{3.4}$$

Additionally, we may bound the norms as:

$$\|\mathbf{e}_{\mathsf{Eval}}\|_\infty \leq O(\ell n \log q)^{O(d)} \cdot \max_{i \in [\ell]} \{\|\mathbf{u}_i - (\mathbf{A}_i + \mathbf{y}[i] \cdot \mathbf{G})^\intercal \mathbf{s}\|_\infty, \ldots\} \tag{3.5}$$

$$\|\mathbf{R}_{\widehat{C} \circ \mathsf{IP}}\|_\infty \leq O(\ell n \log q)^{O(d)} \cdot \max_{i \in [\ell]} \{\|\mathbf{R}_1\|_\infty, \ldots, \|\mathbf{R}_\ell\|_\infty, \|\mathbf{R}'_1\|_\infty, \ldots, \|\mathbf{R}'_t\|_\infty\} \tag{3.6}$$

4 Attacking Predicate Encryption schemes using 1-Keys

In this section, we demonstrate that known LWE based predicate encryption constructions [AFV11, GVW15] are insecure against an adversary that requests 1-keys.

4.1 Attack #1 on [AFV11] using 1-keys.

This attack crucially exploits the stateless nature of the KeyGen algorithm. Since the key generation algorithm is stateless, requesting many keys for the same function results in fresh, independent keys, which may be combined to fully recover the message. We now describe the attack in detail.

Say the attacker requests many keys for the vector $\mathbf{v}$ such that $\langle \mathbf{x}, \mathbf{v} \rangle = 0$ for $b \in \{0, 1\}$. Let $\mathbf{A}_\mathbf{v} = \sum v_i \mathbf{A}_i$. Now by construction of keys in [AFV11], we have:

$$[\mathbf{A} \,|\mathbf{A}_\mathbf{v}] \begin{bmatrix} \mathbf{e}_0 \\ \mathbf{f}_0 \end{bmatrix} = \mathbf{u} \pmod q \tag{4.1}$$

$$[\mathbf{A} \,|\mathbf{A}_\mathbf{v}] \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{f}_1 \end{bmatrix} = \mathbf{u} \pmod q \tag{4.2}$$

$$\text{This implies } [\mathbf{A} \,|\mathbf{A}_\mathbf{v}] \begin{bmatrix} \mathbf{e}_0 - \mathbf{e}_1 \\ \mathbf{f}_0 - \mathbf{f}_1 \end{bmatrix} = \mathbf{0} \pmod q \tag{4.3}$$

Thus, we have a short vector in the lattice $\Lambda_q^\perp(\mathbf{A}|\mathbf{A}_\mathbf{v})$. By making many queries for the same $\mathbf{v}$, the attacker may recover a full trapdoor basis for $\Lambda_q^\perp(\mathbf{A}|\mathbf{A}_\mathbf{v})$. Now, note that the ciphertext contains $\mathbf{A}^\intercal \mathbf{s} + \mathsf{noise}$ as well as $\left(\mathbf{A}_i + \mathbf{x}[i]\mathbf{G} \right)^\intercal \mathbf{s} + \mathsf{noise}$. Since $\langle \mathbf{x}, \mathbf{v} \rangle = 0$, we can follow the decryption procedure to recover $\mathbf{A}_\mathbf{v}^\intercal \mathbf{s} + \mathsf{noise}$ which in turn allows the attacker to recover

$$[\mathbf{A} \,|\mathbf{A}_\mathbf{v}]^\intercal \mathbf{s} + \mathsf{noise}$$

for which he now has a trapdoor. Using the trapdoor, he can now recover the noise terms to get exact linear equations in the LWE secret $\mathbf{s}$, completely breaking LWE security. Note that by functionality, the attacker should only have been able to learn a single bit of information, namely $\langle \mathbf{x}, \mathbf{v} \rangle = 0$.

The reason this attack works in the strong attribute hiding setting, is that a particular linear relation needs to be satisfied to enable decryption, which, given a decrypting key, can be exploited to carry out the attack. Specifically, in the above attack, the decryption procedure allows the attacker to recover $[\mathbf{A} \,|\mathbf{A}_\mathbf{v}]^\intercal \mathbf{s} + \mathsf{noise}$ which would not be possible if the decryption condition did not hold.

4.2 Attack #2 on Partially Hiding Predicate Encryption [GVW15] using 1-keys

This attack exploits the structure of the ciphertext evaluation algorithm in the PHPE construction of [GVW15]. Let

$$\boldsymbol{\beta}_0 = \mathbf{A}^\intercal \mathbf{s} + \mathbf{e}, \quad \boldsymbol{\beta}_1 = \mathbf{P}^\intercal \mathbf{s} + \mathbf{e}' + \mathbf{b}$$
$$\mathbf{c}_1 = \left(\mathbf{A}_1 + \mathbf{y}_1 \cdot \mathbf{G}\right)^\intercal \mathbf{s} + \mathbf{e}_1, \quad \mathbf{c}_2 = \left(\mathbf{A}_2 + \mathbf{y}_2 \cdot \mathbf{G}\right)^\intercal \mathbf{s} + \mathbf{e}_2$$

Then, to compute a product ciphertext corresponding to $\mathbf{y}_1 \cdot \mathbf{y}_2$, the $\mathsf{Eval}_{\mathsf{CT}}$ algorithm does the following:

$$\mathbf{y}_2 \cdot \mathbf{c}_1 = \left(\mathbf{y}_2 \mathbf{A}_1 + \mathbf{y}_1 \mathbf{y}_2 \cdot \mathbf{G}\right)^\intercal \mathbf{s} + \mathbf{y}_2 \cdot \mathbf{e}_1$$
$$\mathbf{G}^{-1}(-\mathbf{A}_1)^\intercal \, \mathbf{c}_2 = \left(\mathbf{A}_2 \mathbf{G}^{-1}(-\mathbf{A}_1) - \mathbf{y}_2 \cdot \mathbf{A}_1\right)^\intercal \mathbf{s} + \mathbf{G}^{-1}(-\mathbf{A}_1)^\intercal \, \mathbf{e}_2$$
$$\mathsf{CT}_{\mathsf{Eval}} = \mathbf{G}^{-1}(-\mathbf{A}_1)^\intercal \, \mathbf{c}_2 + \mathbf{y}_2 \cdot \mathbf{c}_1 = \left(\mathbf{A}_2 \mathbf{G}^{-1}(-\mathbf{A}_1) + \mathbf{y}_1 \mathbf{y}_2 \cdot \mathbf{G}\right)^\intercal \mathbf{s} + \underbrace{\left(\mathbf{G}^{-1}(-\mathbf{A}_1)^\intercal \, \mathbf{e}_2 + \mathbf{y}_2 \cdot \mathbf{e}_1\right)}_{\mathbf{e}_{\mathsf{Eval}}}$$

Now, note that the error term $\mathbf{e}_{\mathsf{Eval}}$ is linear, with known linear coefficients $\mathbf{G}^{-1}(-\mathbf{A}_1)$ and $\mathbf{y}_2$, even though we evaluated a multiplication gate. When decryption succeeds with key $\mathbf{K}$, the decryptor learns

$$\mathbf{e}' - \mathbf{K}^\intercal \cdot \begin{pmatrix} \mathbf{e} \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix} \quad \text{where} \quad \mathbf{e}_{\mathsf{Eval}} = \left(\mathbf{G}^{-1}(-\mathbf{A}_1)^\intercal \, \mathbf{e}_2 + \mathbf{y}_2 \cdot \mathbf{e}_1\right)$$

Thus, a single 1-key (even if it corresponds to a non-linear function) yields a system of m linear equations in the $4m$ variables $\mathbf{e}, \mathbf{e}', \mathbf{e}_1, \mathbf{e}_2$. By requesting another 3 keys [5], the attacker can completely recover the above error terms, which in turn lead to recovery of $\mathbf{s}$, which then permit complete recovery of all the private attributes.

4.3 Attack #3 on Predicate Encryption [GVW15] using 1-keys

Our last attack applies to the construction of Predicate Encryption (PE) from Partially Hiding Predicate Encryption described in [GVW15]. We outline the idea here, and refer the reader to Appendix 8.1 for details. Recall that the PE system of [GVW15] uses (symmetric key) FHE to encrypt the attribute $\mathbf{a}$ and sets the FHE encryption, say $\widehat{\mathbf{a}}$ as the public attribute $\mathbf{y}$ of a PHPE encryption. The FHE secret key $\mathsf{FHE.SK}$ is set to be the PHPE secret attribute $\mathbf{x}$. It then uses PHPE to compute on the public $\widehat{\mathbf{a}}$ to obtain an FHE encryption of the desired function value, say $\widehat{C(\mathbf{a})}$. Next, it uses a PHPE key corresponding to the FHE decryption circuit to compute $\mathsf{FHE.Dec}\left(\widehat{C(\mathbf{a})}, \mathsf{FHE.SK}\right)$.

However, FHE decryption corresponds to threshold inner product, whereas the PHPE construction can only support exact inner product computation while maintaining privacy of the secret attributes. To get around this difficulty, the authors propose the "lazy-OR" trick, which permits the decryptor to learn the exact inner product and compute the threshold on her own. While this is safe if the attacker never gets to decrypt the challenge ciphertext, it leads to an attack if the attacker can get decryption keys. This is because for certain function queries, successful decryption permits the attacker to obtain linear equations (in public coefficients) in the noise terms used in the construction of the FHE ciphertext $\widehat{\mathbf{a}}$. This in turn leads to recovery of the FHE secret, which leads to recovery of the attributes $\mathbf{a}$. For more details, please see Appendix 8.1.

[5] For example, corresponding to $\mathbf{y}_2 \cdot \mathbf{y}_1$, $a_1 \mathbf{y}_1 + a_2 \mathbf{y}_2$ and $b_1 \mathbf{y}_1 + b_2 \mathbf{y}_2$

5 $(1, \text{poly})$-Partially Hiding Predicate Encryption

In the light of the above attacks, it is clear that the $(0, \text{poly})$-PHPE scheme is insecure if the adversary is allowed an unbounded number of 1-keys. In this section, show that the situation can be remedied for the case of a *bounded* number of 1-queries.

Our warm-up construction supports an unbounded number of 0-queries, as in [GVW15], but also a single 1-query. Additionally, the adversary may request the same 1-key an unbounded number of times. We note that previous systems [AFV11, GVW15] were insecure against multiple requests of the same 1-key, see Attack 1 in Section 4. We will show in Section 5.3 how to extend this construction to support any bounded number of 1-queries.

5.1 Construction

The construction of our $(1, \text{poly})$-PHPE scheme is very close to [GVW15]. The main difference is that the randomness used to generate the key for any circuit $\widehat{C} \circ \mathsf{IP}_\gamma$ is chosen using a pseudorandom function $\mathsf{PRF}_{\mathsf{seed}}(\widehat{C} \circ \mathsf{IP}_\gamma)$, so that multiple key requests for the same circuit result in the same key. This modification enables the simulator to sample a single (unique) 1-key $\mathbf{K}^*$ corresponding to some fixed circuit $\widehat{C}^* \circ \mathsf{IP}_\gamma$ such that $\widehat{C}^* \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) = 1$ and mitigates the Attack 1 described in Section 4.

We now proceed to describe the construction.

$\mathsf{PH.Setup}(1^\kappa, 1^t, 1^\ell, 1^d)$: Given as input the security parameter κ, the length of the private and public attributes, t and ℓ respectively, and the depth of the circuit family, do the following:

1. Choose random matrices
$$\mathbf{A}_i \in \mathbb{Z}_q^{n \times m} \text{ for } i \in [\ell],\ \mathbf{B}_i \in \mathbb{Z}_q^{n \times m} \text{ for } i \in [t],\ \mathbf{P} \in \mathbb{Z}_q^{n \times m}$$
To simplify notation, we denote by $\{\mathbf{A}_i\}$ the set $\{\mathbf{A}_i\}_{i \in [\ell]}$ and by $\{\mathbf{B}_i\}$ the set $\{\mathbf{B}_i\}_{i \in [t]}$.
2. Sample $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^m, 1^n, q)$.
3. Let $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ be the powers of two matrix with public trapdoor $\mathbf{T}_{\mathbf{G}}$.
4. Choose a pseudorandom function family $\{\mathsf{PRF}_{\mathsf{seed}}\}_{\mathsf{seed} \in \{0,1\}^\kappa}$ and sample a seed, denoted by seed.
5. Output the public and master secret keys.
$$\mathsf{PH.PK} = \big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \mathbf{A}, \mathbf{P}\big),\ \ \mathsf{PH.MSK} = (\mathsf{PH.PK}, \mathbf{T}, \mathsf{seed})$$

$\mathsf{PH.KeyGen}(\mathsf{PH.MSK}, \widehat{\mathsf{C}} \circ \mathsf{IP}_\gamma)$: Given as input the circuit and the master secret key, do the following:

1. Let $\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} = \mathsf{Eval}_{\mathsf{PK}}\big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \widehat{C} \circ \mathsf{IP}\big)$.
2. Generate randomness rand to be used by the $\mathsf{SampleLeft}$ algorithm using $\mathsf{PRF}_{\mathsf{seed}}(\widehat{C} \circ \mathsf{IP}_\gamma)$.
3. Sample $\mathbf{K}$ such that
$$[\mathbf{A} |\ \mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}] \cdot \mathbf{K} = \mathbf{P} \mod q$$
using $\mathbf{K} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}, \mathbf{T}, \mathbf{P}, s, \mathsf{rand})$. Here rand is the randomness used by the algorithm and s is the standard deviation of the Gaussian being sampled (instantiated in Section 8.2).
4. Output $\mathsf{SK}_{\widehat{C} \circ \mathsf{IP}_\gamma} = \mathbf{K}$.

$\mathsf{PH.Enc}\big(\mathsf{PH.PK}, (\mathbf{x}, \mathbf{y}), \mu\big)$: Given as input the master public key, the private attributes $\mathbf{x}$, public attributes $\mathbf{y}$ and message μ, do the following:

1. Sample $\mathbf{s} \leftarrow \mathcal{D}_{\mathbb{Z}^n, s_B}$ and error terms $\mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m, s_B}$ and $\mathbf{e}' \leftarrow \mathcal{D}_{\mathbb{Z}^m, s_D}$.
2. Let $\mathbf{b} = [0, \ldots, 0, \lceil q/2 \rceil \mu]^\mathsf{T} \in \mathbb{Z}_q^m$. Set
$$\boldsymbol{\beta}_0 = \mathbf{A}^\mathsf{T}\mathbf{s} + \mathbf{e}, \quad \boldsymbol{\beta}_1 = \mathbf{P}^\mathsf{T}\mathbf{s} + \mathbf{e}' + \mathbf{b}$$
3. For $i \in [\ell]$, compute
$$\mathbf{u}_i = (\mathbf{A}_i + \mathbf{y}_i \cdot \mathbf{G})^\mathsf{T}\mathbf{s} + \mathbf{R}_i^\mathsf{T}\mathbf{e}$$
where $\mathbf{R}_i \leftarrow \{-1, 1\}^{m \times m}$.
4. For $i \in [t]$, compute
$$\mathbf{v}_i = (\mathbf{B}_i + \mathbf{x}_i \cdot \mathbf{G})^\mathsf{T}\mathbf{s} + (\mathbf{R}'_i)^\mathsf{T}\mathbf{e}$$
where $\mathbf{R}'_i \leftarrow \{-1, 1\}^{m \times m}$.
5. Output the ciphertext
$$\mathsf{CT}_\mathbf{y} = \big(\mathbf{y}, \boldsymbol{\beta}_0, \boldsymbol{\beta}_1, \{\mathbf{u}_i\}, \{\mathbf{v}_j\}\big)$$
for $i \in [\ell]$, $j \in [t]$.

$\mathsf{PH.Dec}\big(\mathsf{SK}_{\widehat{C} \circ \mathsf{IP}_\gamma}, \mathsf{CT}_\mathbf{y}\big)$: Given as input a secret key and a ciphertext, do the following:

1. Compute
$$\mathbf{u}_{\widehat{C} \circ \mathsf{IP}} = \mathsf{Eval}_{\mathsf{CT}}\big(\{\mathbf{A}_i, \mathbf{u}_i\}, \{\mathbf{B}_j, \mathbf{v}_j\}, \widehat{C} \circ \mathsf{IP}, \mathbf{y}\big)$$
2. Compute
$$\boldsymbol{\nu} = \boldsymbol{\beta}_1 - \mathbf{K}^\mathsf{T} \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} \end{pmatrix}$$
3. Round each coordinate of $\boldsymbol{\nu}$ and if $\big[\mathsf{Round}(\boldsymbol{\nu}[1]), \ldots, \mathsf{Round}(\boldsymbol{\nu}[m-1])\big] = \mathbf{0}$ then set $\mu = \mathsf{Round}(\boldsymbol{\nu}[m])$.
4. Output μ.

Correctness. By the correctness of the $\mathsf{Eval}_{\mathsf{CT}}$ algorithm (Section 3.4.3), we have:

$$\begin{aligned} \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} &= \mathsf{Eval}_{\mathsf{CT}}\big(\{\mathbf{A}_i, \mathbf{u}_i\}, \{\mathbf{B}_i, \mathbf{v}_i\}, \widehat{C} \circ \mathsf{IP}, \mathbf{y}\big) \\ &= (\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \widehat{C} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) \cdot \mathbf{G})^\mathsf{T}\mathbf{s} + \mathbf{e}_{\mathsf{Eval}} \end{aligned}$$

When $\widehat{C} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) = \gamma$, since $\boldsymbol{\beta}_0 = \mathbf{A}^\mathsf{T}\mathbf{s} + \mathbf{e}_0$, we have:

$$\begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} \end{pmatrix} = \begin{pmatrix} \mathbf{A}^\mathsf{T} \\ (\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G})^\mathsf{T} \end{pmatrix} \cdot \mathbf{s} + \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix}$$

Hence,

$$\mathbf{K}^\mathsf{T} \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} \end{pmatrix} \approx \mathbf{P}^\mathsf{T}\mathbf{s} + (\mathbf{K})^\mathsf{T} \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix}$$

$$\text{Thus, } \boldsymbol{\beta}_1 - \mathbf{K}^\mathsf{T} \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} \end{pmatrix} \approx \mathbf{b} + \left\{ \mathbf{e}' - \mathbf{K}^\mathsf{T} \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} \end{pmatrix} \right\}$$

Thus, we require that when $\widehat{C} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) = \gamma$, the first $m-1$ coordinates of the error term $\left\{ \mathbf{e}' - \mathbf{K}^\intercal \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} \end{pmatrix} \right\}$ be bounded above by $q/4$ which can be ensured by our setting of parameters (see Section 8.2).

On the other hand, when $\widehat{C} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) \neq \gamma$, then, letting $\widehat{C} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) = \gamma + \gamma^*$ for some γ^*, and $\mathbf{K}^\intercal = [\mathbf{K}_1^\intercal \,|\mathbf{K}_2^\intercal]$, we have

$$\boldsymbol{\beta}_1 - \mathbf{K}^\intercal \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} \end{pmatrix} \approx \mathbf{b} + \gamma^* \cdot \mathbf{K}_2^\intercal \, \mathbf{G}$$

Thus, the error contains a factor $\gamma^* \cdot \mathbf{K}_2^\intercal \, \mathbf{G}$ which is large. Hence, it holds that the first $m-1$ co-ordinates of the error will be greater than $q/4$ with overwhelming probability, and decryption will output $\perp$, as desired.

5.2 Proof of Security

Next, we argue that the above construction is secure against an adversary who requests an unbounded number of 0-keys and a single 1-key.

Theorem 5.1. *The partially hiding predicate encryption scheme described in Section 5.1 is secure according to definition 3.2.*

Proof. We define a p.p.t. simulator Sim and argue that its output is computationally indistinguishable (under the LWE assumption) from the output of the real world.

Simulator $\mathsf{Sim}\big(1^\kappa, \mathbf{y}, 1^{|\mathbf{x}|}, \widehat{C^*} \circ \mathsf{IP}_\gamma, \mu\big)$: The simulator obtains as input the security parameter, the public attribute $\mathbf{y}$, the size of the private attribute $\mathbf{x}$, the circuit $\widehat{C^*} \circ \mathsf{IP}_\gamma$ such that $\widehat{C^*} \circ \mathsf{IP}_\gamma(\mathbf{x}, \mathbf{y}) = 1$ and μ, which is revealed by correctness of decryption. It does the following:

1. It generates all public parameters as in the real $\mathsf{PH.Setup}$ except $\mathbf{P}$. To generate $\mathbf{P}$, it computes $\mathbf{A}_{\widehat{C^*} \circ \mathsf{IP}} = \mathsf{Eval}_{\mathsf{PK}}\big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \widehat{C^*} \circ \mathsf{IP}\big)$, samples $\mathbf{K}^* \leftarrow \big(\mathcal{D}_{\mathbb{Z}^{2m}, s}\big)^m$ and sets:
$$\mathbf{P} = [\mathbf{A} \mid \mathbf{A}_{\widehat{C^*} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}] \, \mathbf{K}^* \tag{5.1}$$
2. It generates all keys using the real $\mathsf{PH.KeyGen}$ except the key for $\widehat{C^*} \circ \mathsf{IP}_\gamma$, which it sets as $\mathbf{K}^*$ sampled above.
3. For the challenge ciphertext, it samples $\boldsymbol{\beta}_0, \mathbf{u}_i, \mathbf{v}_i$ independently and uniformly from $\mathbb{Z}_q^m$.
4. It computes $\boldsymbol{\beta}_1$ to satisfy the decryption equation corresponding to $\widehat{C^*} \circ \mathsf{IP}_\gamma$ as follows.
   - Let $\mathbf{u}_{\widehat{C^*} \circ \mathsf{IP}} = \mathsf{Eval}_{\mathsf{CT}}\big(\{\mathbf{A}_i, \mathbf{u}_i\}, \{\mathbf{B}_i, \mathbf{v}_i\}, \widehat{C^*} \circ \mathsf{IP}, \mathbf{y}\big)$.
   - Sample $\mathbf{e}'' \leftarrow \mathcal{D}_{\mathbb{Z}^m, s_D}$
   - Set $\boldsymbol{\beta}_1 = (\mathbf{K}^*)^\intercal \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C^*} \circ \mathsf{IP}} \end{pmatrix} + \mathbf{e}'' + \mathbf{b}$ where $\mathbf{b} = [0 \ldots, 0, \lceil q/2 \rceil \mu]^\intercal \in \mathbb{Z}_q^m$.
5. It outputs the challenge ciphertext
$$\mathsf{CT}^* = \Big(\{\mathbf{u}_i\}_{i \in [\ell]}, \{\mathbf{v}_i\}_{i \in [t]}, \mathbf{y}, \boldsymbol{\beta}_0, \boldsymbol{\beta}_1 \Big)$$

We argue that the output of the simulator is distributed indistinguishably from the real world. Intuitively, there are only two differences between the real world and simulated distribution. The first is that instead of choosing $\mathbf{P}$ first and sampling $\mathbf{K}^*$ to satisfy equation 5.1, we now choose $\mathbf{K}^*$ first and set $\mathbf{P}$ accordingly. This is a standard trick in LWE based systems (see the excellent survey [Pei13], where this is trick 1), its first use that we are aware of appears in [GPV08].

The second difference is in how the challenge ciphertext is generated. In our challenge ciphertext the elements $(\boldsymbol{\beta}_0, \mathbf{u}_i, \mathbf{v}_i)$ are sampled uniformly at random while $\boldsymbol{\beta}_1$ which is computed using the elements $(\boldsymbol{\beta}_0, \mathbf{u}_i, \mathbf{v}_i)$ and $\mathbf{K}^*$ in order to satisfy the decryption equation. We note that $\boldsymbol{\beta}_1$ is the only ciphertext element that is generated differently from the challenge ciphertext in the simulator of [GVW15]. In the [GVW15] simulator, $\boldsymbol{\beta}_1$ is also sampled at random, whereas in our case, it is generated to satisfy the decryption equation involving CT^* and $\mathsf{SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)$ since $\widehat{C}^* \circ \mathsf{IP}_\gamma(\mathbf{x}, \mathbf{y}) = 1$. Enforcing this relation is necessary, as it is dictated by the correctness of the system[6].

Formally, we establish the proof via the following hybrids, where the first one corresponds to the real world and the last corresponds to the ideal world.

The Hybrids.

Hybrid 0: The real experiment.

Hybrid 1: The real game algorithm $\mathsf{PH.Setup}$ is replaced with $\mathsf{PH.Setup}_1^*$, which uses the knowledge of $(\mathbf{x}, \mathbf{y}, \widehat{C}^* \circ \mathsf{IP}_\gamma)$ to setup the public parameters and the master key.

Hybrid 2: $\mathsf{PH.Enc}$ is replaced by $\mathsf{PH.Enc}_1^*$, in which $\boldsymbol{\beta}_0$ is computed as in the real world, but all other ciphertext elements are derived from it. The keygen algorithm is as in the real game.

Hybrid 3: The real game $\mathsf{PH.KeyGen}$ is replaced with $\mathsf{PH.KeyGen}_1^*$ where instead of using the trapdoor $\mathbf{T}$ of the matrix $\mathbf{A}$, the secret keys for 0-queries are sampled using the public trapdoor $\mathbf{T_G}$ along with the trapdoor information generated using $\mathsf{PH.Setup}_1^*$ and the secret key for the 1-query $\widehat{C}^* \circ \mathsf{IP}_\gamma$ is sampled using the master key.

Hybrid 4: The encryption algorithm is changed from $\mathsf{PH.Enc}_1^*$ to $\mathsf{PH.Enc}_2^*$. Here, the ciphertext element $\boldsymbol{\beta}_0$ is switched to random and all other ciphertext elements are derived from it.

Hybrid 5: The algorithm $\mathsf{PH.KeyGen}_1^*$ is replaced by $\mathsf{PH.KeyGen}_2^*$. The $\mathsf{PH.KeyGen}_2^*$ algorithm is similar to the real world $\mathsf{PH.KeyGen}$ algorithm, except for the key corresponding to $\widehat{C}^* \circ \mathsf{IP}_\gamma$.

Hybrid 6: The encryption algorithm is changed from $\mathsf{PH.Enc}_2^*$ to $\mathsf{PH.Enc}_3^*$, in which the ciphertext elements $\{\mathbf{u}_i\}$, $\{\mathbf{v}_i\}$ are changed to random. The remaining elements $\boldsymbol{\beta}_0$ and $\boldsymbol{\beta}_1$ are as before.

Hybrid 7: The algorithm $\mathsf{PH.Setup}_1^*$ is replaced by $\mathsf{PH.Setup}_2^*$. The algorithm $\mathsf{PH.Setup}_2^*$ is similar to the real world $\mathsf{PH.Setup}$ algorithm, except for how the matrix $\mathbf{P}$ is generated. This is the simulated world.

We now describe the algorithms used in our hybrids.

[6]Note that the step of "programming" $\boldsymbol{\beta}_1$ forces the simulator to use its knowledge of $\mathbf{y}$. On the other hand, the simulator in [GVW15] does not need to use $\mathbf{y}$ for simulation, implying that even $\mathbf{y}$ is hidden when the attacker does not request 1-keys. Since the real decryption procedure needs $\mathbf{y}$ in order to decrypt, this (in our opinion) further illustrates the weakness of the weak attribute hiding definition.

Auxiliary Algorithms.

$\mathsf{PH.Setup}_1^*(1^\kappa, 1^d, \mathbf{x}, \mathbf{y}, \widehat{C^*} \circ \mathsf{IP}_\gamma)$: Do the following:

1. Sample a matrix with associated trapdoor:

$$(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^m, 1^n, q)$$

   Let $\mathbf{G}$ be the primitive matrix with public trapdoor $\mathbf{T_G}$ that we defined in Section 3.4.1.
2. Let $\mathbf{A}_i = \mathbf{A} \cdot \mathbf{R}_i - \mathbf{y}[i] \cdot \mathbf{G} \in \mathbb{Z}_q^{n \times m}$ for $i \in [\ell]$, where $\mathbf{R}_i \leftarrow \{-1, 1\}^{m \times m}$.
3. Let $\mathbf{B}_i = \mathbf{A} \cdot \mathbf{R}'_i - \mathbf{x}[i] \cdot \mathbf{G} \in \mathbb{Z}_q^{n \times m}$ for $i \in [t]$, where $\mathbf{R}'_i \leftarrow \{-1, 1\}^{m \times m}$.
4. Compute $\mathbf{R}_{\widehat{C^*} \circ \mathsf{IP}} \leftarrow \mathsf{Eval_R}\big(\{\mathbf{R}_i\},\ \{\mathbf{R}'_i\},\ \{\mathbf{B}_i\},\ \mathbf{y},\ \widehat{C^*} \circ \mathsf{IP}\big)$ algorithm.
5. Sample $\mathbf{K}^* \leftarrow \big(\mathcal{D}_{\mathbb{Z}^{2m}, s}\big)^m$ and set

$$\mathbf{P} = [\mathbf{A} \mid \mathbf{A} \cdot \mathbf{R}_{\widehat{C^*} \circ \mathsf{IP}}] \cdot \mathbf{K}^*$$

   Note that

$$\text{If} \quad \mathbf{A}_{\widehat{C^*} \circ \mathsf{IP}} \leftarrow \mathsf{Eval_{PK}}\big(\{\mathbf{A}_i\},\ \{\mathbf{B}_j\},\ \widehat{C^*} \circ \mathsf{IP}\big), \quad \text{and} \quad \widehat{C^*} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) = \gamma,$$
$$\text{then,} \quad [\mathbf{A}_{\widehat{C^*} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}] = [\mathbf{A} \cdot \mathbf{R}_{\widehat{C^*} \circ \mathsf{IP}}]$$

6. Output the master public key as $\mathsf{PH.PK} = \big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \mathbf{A}, \mathbf{P}\big)$ and the master secret key as $\mathsf{PH.MSK} = \big(\mathbf{T}, \{\mathbf{R}_i\}, \{\mathbf{R}'_i\}, \mathbf{K}^*\big)$.

$\mathsf{PH.Enc}_1^*\big(\ \mathsf{PH.PK}, \mathsf{PH.MSK}, \mathbf{y}, \mu, \widehat{C^*} \circ \mathsf{IP}\ \big)$: The ciphertext is computed as follows:

1. Sample $\mathbf{s}, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^n, s_B}$ and set $\boldsymbol{\beta}_0 = \mathbf{A}^\mathsf{T}\mathbf{s} + \mathbf{e}$.
2. For $i \in [\ell]$, $j \in [t]$ compute

$$\mathbf{u}_i = \mathbf{R}_i^\mathsf{T} \cdot \boldsymbol{\beta}_0, \quad \mathbf{v}_j = (\mathbf{R}'_j)^\mathsf{T} \cdot \boldsymbol{\beta}_0$$

3. Compute $\boldsymbol{\beta}_1$ as follows.
   - Compute $\mathbf{u}_{\widehat{C^*} \circ \mathsf{IP}} = \mathsf{Eval_{CT}}\big(\{\mathbf{A}_i, \mathbf{u}_i\}, \{\mathbf{B}_i, \mathbf{v}_i\}, \widehat{C^*} \circ \mathsf{IP}, \mathbf{y}\big)$.
   - Sample $\mathbf{e}'' \leftarrow \mathcal{D}_{\mathbb{Z}^m, s_D}$.
   - Set $\boldsymbol{\beta}_1 = (\mathbf{K}^*)^\mathsf{T} \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C^*} \circ \mathsf{IP}} \end{pmatrix} + \mathbf{e}'' + \mathbf{b}$ where $\mathbf{b} = [0 \ldots, 0, \lceil q/2 \rceil \mu]^\mathsf{T} \in \mathbb{Z}_q^m$.
4. Output $\Big(\{\mathbf{u}_i\}, \{\mathbf{v}_i\}, \mathbf{y}, \boldsymbol{\beta}_0, \boldsymbol{\beta}_1\Big)$

$\mathsf{PH.KeyGen}_1^*(\mathsf{PH.MSK}, \widehat{C} \circ \mathsf{IP}_\rho, \widehat{C^*} \circ \mathsf{IP}_\gamma)$: Do the following:

1. Compute the homomorphic public key corresponding to the circuit $\widehat{C} \circ \mathsf{IP}$ as

$$\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} \leftarrow \mathsf{Eval_{PK}}\big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \widehat{C} \circ \mathsf{IP}\big)$$

2. By section 3.4.3, we have that $\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} = \mathbf{A} \cdot \mathbf{R}_{\widehat{C} \circ \mathsf{IP}} - \langle \mathbf{x}, \widehat{C}(\mathbf{y}) \rangle \cdot \mathbf{G}$. An admissible adversary can request:

- A circuit $\widehat{C} \circ \mathsf{IP}_\rho$ such that $\langle \mathbf{x}, \widehat{C}(\mathbf{y}) \rangle \neq \rho$, hence $\widehat{C} \circ \mathsf{IP}_\rho(\mathbf{x}, \mathbf{y}) = 0$. In this case, we have:

$$[\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \rho \cdot \mathbf{G}] = [\mathbf{A} \cdot \mathbf{R}_{\widehat{C} \circ \mathsf{IP}} + (\rho - \langle \mathbf{x}, \widehat{C}(\mathbf{y}) \rangle) \cdot \mathbf{G}]$$

  Hence, we may sample $\mathbf{K} \leftarrow \mathsf{SampleRight}\big(\mathbf{A},\ (\rho - \langle \mathbf{x}, \widehat{C}(\mathbf{y}) \rangle) \cdot \mathbf{G},\ \mathbf{R}_{\widehat{C} \circ \mathsf{IP}},\ \mathbf{T}_{\mathbf{G}},\ \mathbf{P},\ s,\ \mathsf{rand}\big)$ so that:

$$[\mathbf{A} \mid \mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \rho \cdot \mathbf{G}]\ \mathbf{K} = \mathbf{P} \pmod q$$

  Note that rand is the randomness used by the $\mathsf{SampleRight}$ algorithm, generated using $\mathsf{PRF}_{\mathsf{seed}}(\widehat{C} \circ \mathsf{IP}_\rho)$, so that multiple requests for $\widehat{C} \circ \mathsf{IP}_\rho$ result in the same key.
- The circuit $\widehat{C}^* \circ \mathsf{IP}_\gamma$, such that $\langle \mathbf{x}, \widehat{C}^*(\mathbf{y}) \rangle = \gamma$ in which case we let $\mathbf{K} = \mathbf{K}^*$. Note that,

$$[\mathbf{A}_{\widehat{C}^* \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}] = [\mathbf{A} \cdot \mathbf{R}_{\widehat{C}^* \circ \mathsf{IP}}]$$
$$\text{Hence, } [\ \mathbf{A} \mid \mathbf{A}_{\widehat{C}^* \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}\ ]\ \mathbf{K}^* = \mathbf{P} \pmod q$$

3. Return $\mathsf{SK}_{\widehat{C} \circ \mathsf{IP}_\rho} = \mathbf{K}$.

$\mathsf{PH.Enc}_2^*(\mathsf{PH.PK}, \mathsf{PH.MSK}, \mathbf{y}, \widehat{C^*} \circ \mathsf{IP})$: Sample $\boldsymbol{\beta}_0 \leftarrow \mathbb{Z}_q^m$ randomly. Compute the remaining ciphertext elements as in $\mathsf{PH.Enc}_1^*$.

$\mathsf{PH.KeyGen}_2^*(\mathsf{PH.PK}, \mathsf{PH.MSK}, \widehat{C} \circ \mathsf{IP}_\rho, \widehat{C}^* \circ \mathsf{IP}_\gamma)$: Do the following:

1. If $\widehat{C} \circ \mathsf{IP}_\rho \neq \widehat{C}^* \circ \mathsf{IP}_\gamma$, sample the key $\mathbf{K}$ using the $\mathsf{SampleLeft}$ algorithm as in KeyGen.
2. If $\widehat{C} \circ \mathsf{IP}_\rho = \widehat{C}^* \circ \mathsf{IP}_\gamma$, let $\mathbf{K} = \mathbf{K}^*$.
3. Return $\mathsf{SK}_{\widehat{C} \circ \mathsf{IP}_\rho} = \mathbf{K}$.

$\mathsf{PH.Enc}_3^*(\mathsf{PH.PK}, \mathsf{PH.MSK}, \mathbf{y}, \widehat{C^*} \circ \mathsf{IP})$: Sample $\mathbf{u}_i$, $\mathbf{v}_i$ randomly – all other terms are computed as in $\mathsf{PH.Enc}_2^*$.

$\mathsf{PH.Setup}_2^*(1^\kappa, 1^d, \widehat{C}^* \circ \mathsf{IP}_\gamma)$: Do the following:

1. Sample the parameters $(\mathbf{A}, \{\mathbf{A}_i\}, \{\mathbf{B}_i\})$ as in $\mathsf{PH.Setup}$.
2. Compute $\mathbf{A}_{\widehat{C}^* \circ \mathsf{IP}} \leftarrow \mathsf{Eval}_{\mathsf{PK}}\big(\{\mathbf{A}_i\},\ \{\mathbf{B}_i\},\ \widehat{C^*} \circ \mathsf{IP}\big)$.
3. Sample $\mathbf{K}^* \leftarrow \big(\mathcal{D}_{\mathbb{Z}^{2m}, \sigma}\big)^m$ and set

$$\mathbf{P} = [\mathbf{A} \mid \mathbf{A}_{\widehat{C^*} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}] \cdot \mathbf{K}^*$$

4. Output the master public key as $\mathsf{PH.PK} = \big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \mathbf{A}, \mathbf{P}\big)$ and the master secret key as $\mathsf{PH.MSK} = \big(\mathbf{T}, \mathbf{K}^*\big)$.

Indistinguishability of Hybrids. Below we argue that consecutive hybrids are indistinguishable.

Lemma 5.2. *Hybrid 0 and Hybrid 1 are statistically indistinguishable.*

Proof. The only difference between the two hybrids is in how the public parameters are generated. In Hybrid 0,

$$\mathbf{A} \leftarrow \mathsf{TrapGen}(1^m, 1^n, q), \quad \{\mathbf{A}_i\},\ \{\mathbf{B}_j\},\ \mathbf{P} \leftarrow \mathbb{Z}_q^{n \times m} \quad \forall i \in [\ell], j \in [t]$$

By correctness of $\mathsf{TrapGen}$ algorithm (see Section 3.4.1), we have that in Hybrid 0,

$$\Big(\mathbf{A},\ \{\mathbf{A}_i\},\ \{\mathbf{B}_j\},\ \mathbf{P} \Big) \overset{s}{\approx} \Big(\mathcal{U},\ \{\mathcal{U}\},\ \{\mathcal{U}\},\ \mathcal{U} \Big)$$

where $\mathcal{U}$ denotes the uniform distribution over $\mathbb{Z}_q^{n\times m}$.

In Hybrid 1, we have:

$$\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m},\quad \mathbf{A}_i = \mathbf{A}\cdot\mathbf{R}_i - \mathbf{y}[i]\cdot\mathbf{G},\quad \mathbf{B}_j = \mathbf{A}\cdot\mathbf{R}'_j - \mathbf{x}[j]\cdot\mathbf{G},\quad \mathbf{P} = [\mathbf{A} \mid \mathbf{A}\cdot\mathbf{R}_{\widehat{C^*\circ\mathsf{IP}}}]\cdot\mathbf{K}^*$$

where $\mathbf{R}_i, \mathbf{R}'_i \leftarrow \{-1,1\}^{m\times m}$, $\mathbf{K}^* \in \left(\mathcal{D}_{\mathbb{Z}^{2m},\sigma}\right)^m$. $\mathbf{A}$ is chosen uniformly in Hybrid 2.

$$\text{Let}\quad \mathbf{K}^* = \begin{pmatrix} \mathbf{K}_1^* \\ \mathbf{K}_2^* \end{pmatrix}.\ \text{Then}\quad \mathbf{P} = \mathbf{P}_1 + \mathbf{P}_2$$
$$\text{where } \mathbf{P}_1 = \mathbf{A}\cdot\mathbf{K}_1^*\quad\text{and}\quad \mathbf{P}_2 = \mathbf{A}\cdot\mathbf{R}_{\widehat{C^*\circ\mathsf{IP}}}\cdot\mathbf{K}_2^*$$

By Section 3.4.2 and [GPV08, Lemma 5.2], we have $(\mathbf{A},\mathbf{P}_1) \overset{s}{\approx} (\mathbf{A},\mathcal{U})$. Now, consider the distribution of Hybrid 2:

$$\begin{aligned}
\Big(\mathbf{A},\ \{\mathbf{A}_i\},\ \{\mathbf{B}_j\},\ \mathbf{P}_1+\mathbf{P}_2 \Big) &\overset{s}{\approx} \Big(\mathbf{A},\ \{\mathbf{A}_i\},\ \{\mathbf{B}_j\},\ \mathcal{U} \Big) \\
&\overset{s}{\approx} \Big(\mathbf{A},\ \{\mathcal{U}\},\ \{\mathcal{U}\},\ \mathcal{U} \Big) \text{ by [GVW15, } \textit{Lemma 2.2}\text{]} \\
&\overset{s}{\approx} \Big(\mathcal{U},\ \{\mathcal{U}\},\ \{\mathcal{U}\},\ \mathcal{U} \Big)
\end{aligned}$$

Since this is the distribution in Hybrid 0, we are done. □

Lemma 5.3. *Hybrid 1 and Hybrid 2 are statistically indistinguishable.*

Proof. The difference between the two hybrids is in how the ciphertext elements $\{\mathbf{u}_i\}$, $\{\mathbf{v}_j\}$, $\boldsymbol{\beta}_1$ are generated. In Hybrid 1, for $i \in [\ell]$, $j \in [t]$, we have

$$\mathbf{u}_i = (\mathbf{A}_i + \mathbf{y}[i]\cdot\mathbf{G})^\intercal\mathbf{s} + \mathbf{R}_i^\intercal\mathbf{e},\quad \mathbf{v}_j = (\mathbf{B}_j + \mathbf{x}[j]\cdot\mathbf{G})^\intercal\mathbf{s} + (\mathbf{R}'_j)^\intercal\mathbf{e},\quad \boldsymbol{\beta}_1 = \mathbf{P}^\intercal\mathbf{s} + \mathbf{e}_1 + \mathbf{b}$$

By construction of $\mathbf{P}$, we may write:

$$\begin{aligned}
\boldsymbol{\beta}_1 &= \mathbf{P}^\intercal\mathbf{s} + \mathbf{e}_1 + \mathbf{b} \\
&= (\mathbf{K}^*)^\intercal \begin{pmatrix} \mathbf{A}^\intercal \\ \mathbf{R}_{\widehat{C^*\circ\mathsf{IP}}}^\intercal\mathbf{A}^\intercal \end{pmatrix}\cdot\mathbf{s} + \mathbf{e}_1 + \mathbf{b} \quad\in\ \mathbb{Z}_q^m
\end{aligned}$$

In Hybrid 2, for $i \in [\ell]$, $j \in [t]$:

$$\mathbf{u}_i = \mathbf{R}_i^\intercal\cdot\boldsymbol{\beta}_0,\quad \mathbf{v}_j = (\mathbf{R}')_j^\intercal\cdot\boldsymbol{\beta}_0,\quad \boldsymbol{\beta}_1 = (\mathbf{K}^*)^\intercal \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C^*\circ\mathsf{IP}}} \end{pmatrix} + \mathbf{e}'' + \mathbf{b}$$

Now, since $\boldsymbol{\beta}_0 = \mathbf{A}^\intercal\mathbf{s} + \mathbf{e}$, we have,

$$\begin{aligned}
\mathbf{R}_i^\intercal\boldsymbol{\beta}_0 &= (\mathbf{A}\mathbf{R}_i)^\intercal\mathbf{s} + \mathbf{R}_i^\intercal\mathbf{e} \\
&= (\mathbf{A}_i + \mathbf{y}[i]\cdot\mathbf{G})^\intercal\mathbf{s} + \mathbf{R}_i^\intercal\mathbf{e} \\
\text{Similarly, we have, } (\mathbf{R}'_j)^\intercal\boldsymbol{\beta}_0 &= (\mathbf{B}_j + \mathbf{x}[j]\cdot\mathbf{G})^\intercal\mathbf{s} + (\mathbf{R}'_j)^\intercal\mathbf{e}
\end{aligned}$$

as in Hybrid 1. Next, consider the distribution of $\boldsymbol{\beta}_1$. Recall that

$$\begin{aligned}
\mathbf{u}_{\widehat{C^*\circ\mathsf{IP}}} &= \mathsf{Eval}_{\mathsf{CT}}\big(\{\mathbf{A}_i, \mathbf{u}_i\}, \{\mathbf{B}_i, \mathbf{v}_i\}, \widehat{C^*} \circ \mathsf{IP}, \mathbf{y}\big) \\
&= \big(\mathbf{A}_{\widehat{C^*\circ\mathsf{IP}}} + \widehat{C^*} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) \cdot \mathbf{G}\big)^{\mathsf{T}}\mathbf{s} + \mathbf{e}_{\mathsf{Eval}} \quad \text{by definition of } \mathsf{Eval}_{\mathsf{CT}} \\
&= \big(\mathbf{A}\mathbf{R}_{\widehat{C^*\circ\mathsf{IP}}} + (\langle \mathbf{x}, \widehat{C^*}(\mathbf{y})\rangle - \langle \mathbf{x}, \widehat{C^*}(\mathbf{y})\rangle) \cdot \mathbf{G}\big)^{\mathsf{T}}\mathbf{s} + \mathbf{e}_{\mathsf{Eval}} \\
&= \big(\mathbf{A}\mathbf{R}_{\widehat{C^*\circ\mathsf{IP}}}\big)^{\mathsf{T}}\mathbf{s} + \mathbf{e}_{\mathsf{Eval}}
\end{aligned}$$

Since $\boldsymbol{\beta}_0 = \mathbf{A}^{\mathsf{T}}\mathbf{s} + \mathbf{e}_0$, we have:

$$\begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C^*\circ\mathsf{IP}}} \end{pmatrix} = \begin{pmatrix} \mathbf{A}^{\mathsf{T}} \\ \mathbf{R}^{\mathsf{T}}_{\widehat{C^*\circ\mathsf{IP}}}\mathbf{A}^{\mathsf{T}} \end{pmatrix} \cdot \mathbf{s} + \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix}$$

Hence,

$$\begin{aligned}
\boldsymbol{\beta}_1 &= (\mathbf{K}^*)^{\mathsf{T}}\left\{ \begin{pmatrix} \mathbf{A}^{\mathsf{T}} \\ \mathbf{R}^{\mathsf{T}}_{\widehat{C^*\circ\mathsf{IP}}}\mathbf{A}^{\mathsf{T}} \end{pmatrix} \cdot \mathbf{s} + \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix} \right\} + \mathbf{e}'' + \mathbf{b} \\
&= (\mathbf{K}^*)^{\mathsf{T}} \begin{pmatrix} \mathbf{A}^{\mathsf{T}} \\ \mathbf{R}^{\mathsf{T}}_{\widehat{C^*\circ\mathsf{IP}}}\mathbf{A}^{\mathsf{T}} \end{pmatrix} \cdot \mathbf{s} + \left\{ (\mathbf{K}^*)^{\mathsf{T}} \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix} + \mathbf{e}'' \right\} + \mathbf{b}
\end{aligned}$$

Thus, it suffices to ensure that

$$\mathbf{e}_1 \overset{s}{\approx} \left\{ (\mathbf{K}^*)^{\mathsf{T}} \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix} + \mathbf{e}'' \right\} \tag{5.2}$$

where the LHS is the error used in construction of $\boldsymbol{\beta}_1$ in Hybrid 1, and the RHS is the error used in construction of $\boldsymbol{\beta}_1$ in Hybrid 2. This may be ensured by choosing $\mathbf{e}_1$ and $\mathbf{e}''$ to be sufficiently larger than $(\mathbf{K}^*)^{\mathsf{T}} \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix}$ so that Equation 5.2 is satisfied. Please see Section 8.2 for details about parameters. □

Lemma 5.4. *Hybrid 2 and Hybrid 3 are statistically indistinguishable.*

Proof. Between Hybrid 2 and Hybrid 3, the manner of generating keys changes from $\mathsf{PH.KeyGen}$ to $\mathsf{PH.KeyGen}^*_1$. We must consider two cases:

1. For 0-keys, the argument is exactly the same as in [GVW15, Lemma 3.6]. In Hybrid 2, these keys are sampled using the $\mathsf{SampleLeft}$ algorithm, whereas in Hybrid 3, they are sampled using the $\mathsf{SampleRight}$ algorithm. By Section 3.4.2, the resultant distribution is the same.

2. For 1-keys, first note that since $\widehat{C^*} \circ \mathsf{IP}(\mathbf{x}, \mathbf{y}) = \gamma$, and we have $\mathbf{A}_{\widehat{C^*\circ\mathsf{IP}}} = \mathbf{A}\mathbf{R}_{\widehat{C^*\circ\mathsf{IP}}} - \gamma \cdot \mathbf{G}$ by Section 3.4.3, it holds that:

$$[\mathbf{A} \mid \mathbf{A}_{\widehat{C^*\circ\mathsf{IP}}} + \gamma \cdot \mathbf{G}] = [\mathbf{A} \mid \mathbf{A}\mathbf{R}_{\widehat{C^*\circ\mathsf{IP}}}]$$

   Next, note that $\mathbf{P}$ is chosen the same way in both Hybrid 2 and Hybrid 3, namely, sample $\mathbf{K}^* \leftarrow \left(\mathcal{D}_{\mathbb{Z}^{2m},s}\right)^m$ and set:

$$\mathbf{P} = [\mathbf{A} \mid \mathbf{A}\mathbf{R}_{\widehat{C^*\circ\mathsf{IP}}}]\, \mathbf{K}^*$$

Let $\mathbf{F} = [\mathbf{A} \mid \mathbf{AR}_{\widehat{C}^* \circ \mathsf{IP}}]$. Now, in Hybrid 2, given $(\mathbf{F}, \mathbf{P})$, we sample $\mathbf{K}$ using $\mathsf{SampleLeft}$ so that

$$[\mathbf{A} \mid \mathbf{AR}_{\widehat{C}^* \circ \mathsf{IP}}]\, \mathbf{K} = \mathbf{P} \mod q \tag{5.3}$$

By Section 3.4.2, the distribution of $\mathbf{K}$ is statistically close to $\left(\mathcal{D}_{\Lambda_q^{\mathbf{P}}(\mathbf{F}),s}\right)^m$.

In Hybrid 3, we output $\mathbf{K}^*$ which was sampled to generate $\mathbf{P}$ in $\mathsf{PH.Setup}_1^*$. Given $(\mathbf{F}, \mathbf{P})$ we have by Section 3.4.2, that $\mathbf{K}^*$ is distributed as $\left(\mathcal{D}_{\Lambda_q^{\mathbf{P}}(\mathbf{F}),s}\right)^m$.

Note that in Hybrid 2, the $\mathsf{SampleLeft}$ algorithm is invoked using randomness generated by a PRF, hence repeated requests for the circuit $\widehat{C}^* \circ \mathsf{IP}$ result in the same key. This matches with the simulation, which can only know a single solution to Equation 5.3.

Hence, the distribution of the 1-key for $\widehat{C}^* \circ \mathsf{IP}$ in Hybrids 2 and 3 is equivalent.

□

Lemma 5.5. *Hybrid 3 and Hybrid 4 are computationally indistinguishable under the* LWE *assumption.*

Proof. As in [GVW15], we show how the LWE assumption can be broken given an adversary that distinguishes between Hybrids 3 and 4. Given the sample $(\mathbf{A}, \mathbf{u})$ where $\mathbf{u}$ is either real or random, the remainder of the distribution can be sampled as follows:

1. Run $\mathsf{PH.Setup}_1^*$ and $\mathsf{PH.KeyGen}_1^*$ using $\mathbf{A}$ that is provided. These algorithms are used to produce the public parameters and the function keys.
2. To produce the ciphertext, set $\boldsymbol{\beta}_0 = \mathbf{u}$ and generate the remaining components as in Hybrid 3 (and also Hybrid 2), namely,
$$\mathbf{u}_i = \mathbf{R}_i^\intercal \boldsymbol{\beta}_0, \quad \mathbf{v}_i = \mathbf{R'}_i^\intercal \boldsymbol{\beta}_0$$
and
$$\boldsymbol{\beta}_1 = (\mathbf{K}^*)^\intercal \begin{pmatrix} \boldsymbol{\beta}_0 \\ \mathbf{u}_{\widehat{C^*} \circ \mathsf{IP}} \end{pmatrix} + \mathbf{e}'' + \mathbf{b}$$

Now if $\mathbf{u} = \mathbf{A}^\intercal \mathbf{s} + \mathbf{e}$ then we get the distribution of Hybrid 3, while if $\mathbf{u}$ is real and of Hybrid 4 if $\mathbf{u}$ is random. □

Lemma 5.6. *Hybrid 4 and Hybrid 5 are statistically indistinguishable.*

Proof. The proof is analogous to the proof of indistinguishability between Hybrids 2 and 3. □

Lemma 5.7. *Hybrid 5 and Hybrid 6 are statistically indistinguishable.*

Proof. The only difference between Hybrids 5 and 6 is in how $\mathbf{u}_i, \mathbf{v}_i$ are generated. To argue indistinguishability, we must show:

$$\left(\mathbf{A}, \mathbf{B}, \boldsymbol{\beta}_0, \{\mathbf{AR}_i, \mathbf{R}_i^\intercal \boldsymbol{\beta}_0\}, \{\mathbf{BR}'_i, (\mathbf{R}'_i)^\intercal \boldsymbol{\beta}_0\}\right) \approx \left(\mathbf{A}, \mathbf{B}, \boldsymbol{\beta}_0, \{\mathbf{AR}_i, \mathbf{u}_i\}, \{\mathbf{BR}'_i, \mathbf{v}_i\}\right)$$

This follows directly from [GVW15, Lemma 3.9]. Note that $\boldsymbol{\beta}_1$ is generated from the above components in the exact same way in both Hybrids, and hence does not feature in the joint distribution.

□

Lemma 5.8. *Hybrid 6 and Hybrid 7 are statistically indistinguishable.*

Proof. The proof follows similarly as the proof of indistinguishability between Hybrids 0 and 1. □

□

Thus, we have shown that the real world and simulated world are indistinguishable. We refer the reader to Appendix 8.2, for the setting of parameters.

5.3 (Q, poly)-Partially Hiding Predicate Encryption

In this section, we describe how to generalise the above scheme to handle Q queries. As discussed in Section 1, we repurpose a trick suggested by Gorbunov et al. [GVW12]. The idea is to enable the key generator to generate a fresh matrix $\mathbf{P}_i^*$ for the i^{th} key in a stateless manner. To do this, we publish a set of matrices $\{\mathbf{P}_1, \ldots, \mathbf{P}_k\}$ in the public key, where k is a parameter to be chosen later. The key generator chooses a random subset $\Delta_i \subset [k]$ s.t. $|\Delta_i| = v$ for some suitably chosen v, and computes $\mathbf{P}_i^* = \sum_{j \in \Delta_i} \mathbf{P}_j$. It then samples $\mathbf{K}_i$ so that

$$[\ \mathbf{A} \mid \mathbf{A}_{C_i}\ ]\ \mathbf{K}_i = \mathbf{P}_i^* \mod q$$

If we choose (v, k) as functions of parameters of (κ, Q) in a way that the Q subsets $\Delta_1, \ldots, \Delta_Q$ are cover free with high probability, then this ensures that the matrices $\mathbf{P}_1^*, \ldots, \mathbf{P}_Q^*$ are independent and uniformly distributed, which will enable the simulator to sample the requisite keys. It suffices to choose $v = \Theta(\kappa)$ and $k = \Theta(v \cdot Q^2)$ for this to happen (see [GVW12, Section 5.2] for more details).

Below, we describe the strategy more formally. For ease of exposition, we only outline the differences from the $(1, \text{poly})$-scheme described above.

1. $\mathsf{QPH.Setup}$**.** The setup algorithm is modified so that instead of choosing a single $\mathbf{P}$ it now chooses a set $\mathbf{P}_1, \ldots, \mathbf{P}_k \leftarrow \mathbb{Z}_q^{n \times m}$.

2. $\mathsf{QPH.Enc}$**.** The encryptor computes $\big\{\{\mathbf{u}_i\}, \{\mathbf{v}_j\}, \boldsymbol{\beta}_0\big\}$ as before, but instead of computing a single $\boldsymbol{\beta}_1$, it now computes k encodings of the message as:

$$\{\ \boldsymbol{\beta}_{1j} = \mathbf{P}_j^\intercal\ \mathbf{s} + \mathbf{e}'_j + \mathbf{b}\ \}_{j \in [k]}$$

3. $\mathsf{QPH.KeyGen}$**.** To produce a key for a circuit $\widehat{C} \circ \mathsf{IP}$, the key generator samples a random binary vector $\mathbf{r}$ with hamming weight v and computes the subset sum $\mathbf{P}^* = \sum_{j \in [k]} r_j\ \mathbf{P}_j$. It now produces a key $(\mathbf{K}, \mathbf{r})$ where $\mathbf{K}$ is sampled using $\mathsf{SampleLeft}$ to satisfy:

$$[\mathbf{A}|\ \mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}]\ \mathbf{K} = \mathbf{P}^*$$

4. $\mathsf{QPH.Dec}$**.** Given the ciphertext $\{\mathbf{u}_i\}, \{\mathbf{v}_j\}, \boldsymbol{\beta}_0, \{\boldsymbol{\beta}_{1j}\}$ and a key $(\mathbf{K}, \mathbf{r})$, the decryptor computes

$$\begin{aligned}
\boldsymbol{\beta}_1^* &= \sum_{j \in [k]} r_j\ \boldsymbol{\beta}_{1j} \\
&= \sum_{j \in [k]} r_j\ \left(\mathbf{P}_j^\intercal\ \mathbf{s} + \mathbf{e}'_j + \mathbf{b}\right) \\
&= (\mathbf{P}^*)^\intercal \mathbf{s} + \mathbf{e}_{\mathbf{r}} + \big(\ \sum_{j \in [k]} r_j\ \big)\mathbf{b} \quad \text{where } \mathbf{e}_{\mathbf{r}} = \sum_{j \in [k]} r_j\ \mathbf{e}'_j
\end{aligned}$$

By correctness of the $(1, \text{poly})$-scheme, this lets the decryptor learn $\sum_{j\in[k]} r_j\ \mathbf{b}$, which lets her recover $\mathbf{b}$ since $\mathbf{r}$ is provided.

Generalizing the proof of security. We describe how to generalize the simulator. The simulator receives as input the security parameter, the public attribute $\mathbf{y}$, the size of the private attribute $\mathbf{x}$, Q circuits $\{\widehat{C_i^*} \circ \mathsf{IP}_{\gamma_i}\}_{i\in[Q]}$ such that $\widehat{C_i^*} \circ \mathsf{IP}_{\gamma_i}(\mathbf{x}, \mathbf{y}) = 1$, and μ, which is revealed by correctness of decryption. It does the following:

1. **Generating the Public Key.** It generates the public parameters $\{\mathbf{B}_1, \ldots, \mathbf{B}_t\}$, $\mathbf{A}$ as in the real $\mathsf{PH.Setup}$. For $\{\mathbf{A}_1, \ldots, \mathbf{A}_\ell\}$, it samples $\mathbf{R}_i \leftarrow \{-1, 1\}^{m\times m}$ and sets $\mathbf{A}_i = \mathbf{A}\ \mathbf{R}_i - \mathbf{y}[i] \cdot \mathbf{G}$ for $i \in [\ell]$. Note that the simulator knows $\mathbf{y}$.

   It generates $\mathbf{P}_1, \ldots, \mathbf{P}_k$ as follows:

   (a) For $i \in [Q]$ compute

   $$\mathbf{A}_{\widehat{C_i^*}\circ\mathsf{IP}} = \mathsf{Eval}_{\mathsf{PK}}\big(\{\mathbf{A}_j\}_{j\in[\ell]}, \{\mathbf{B}_j\}_{j\in[t]}, \widehat{C_i^*} \circ \mathsf{IP}\big)$$

   (b) Next, sample $\mathbf{K}_i^* \leftarrow \big(\mathcal{D}_{\mathbb{Z}^{2m},s}\big)^m$ and set:

   $$\begin{aligned}\mathbf{P}_i^* &= [\mathbf{A} \mid \mathbf{A}_{\widehat{C_i^*}\circ\mathsf{IP}} + \gamma_i \cdot \mathbf{G}]\ \mathbf{K}_i^* \quad \forall i \in [Q] \\ &= [\mathbf{A} \mid \mathbf{A}\mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}}]\ \mathbf{K}_i^*\end{aligned}$$

   where $\mathbf{K}_i^* = \begin{pmatrix} \mathbf{K}_{1i}^* \\ \mathbf{K}_{2i}^* \end{pmatrix}$.

   (c) Choose Q random binary vectors $\mathbf{r}_1, \ldots, \mathbf{r}_Q \in \{0, 1\}^k$ of hamming weight v.

   (d) For $i \in [Q]$, write down the following equations in variables $\mathbf{K}_1, \ldots, \mathbf{K}_k$:

   $$\mathbf{A}\ \mathbf{K}_{1i}^* + \mathbf{A}\ \mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}}\ \mathbf{K}_{2i}^* = \mathbf{A}\ \Big(\sum_{j\in[k]} \mathbf{r}_i[j]\ \mathbf{K}_j\Big) \tag{5.4}$$

   Now, to satisfy the above, it suffices to set:

   $$\mathbf{K}_{1i}^* + \mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}}\ \mathbf{K}_{2i}^* = \sum_{j\in[k]} \mathbf{r}_i[j]\ \mathbf{K}_j \tag{5.5}$$

   (e) Next, note that by cover-freeness, we have that for each equation $i \in [Q]$, the subset sum in the RHS contains at least one matrix, say $\mathbf{K}_{i'}$, which does not appear in any other equation. Rearranging terms, we get for $i \in [Q]$,

   $$\mathbf{K}_{i'} = \Big(\mathbf{K}_{1i}^* + \mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}}\ \mathbf{K}_{2i}^*\Big) - \Big(\sum_{j:j\neq i'} \mathbf{r}_i[j]\ \mathbf{K}_j\Big) \tag{5.6}$$

   (f) Sample all the $\mathbf{K}_j$ from $\mathcal{D}_{\mathbb{Z}^m,s}$ that appear in the RHS of the above equations, namely $\mathbf{K}_j$ where $j \neq i'$, $j \in [k]$, $i' \in [Q]$. Set the Q values $\mathbf{K}_{i'}$ in the LHS to satisfy the above equation.

   (g) For $i \in [k]$, set $\mathbf{P}_i = \mathbf{A}\ \mathbf{K}_i \mod q$.

2. **Generating Function Keys.** It generates all keys using the real $\mathsf{PH.KeyGen}$ except the Q keys $\{\widehat{C_i^*} \circ \mathsf{IP}_{\gamma_i}\}_{i\in[Q]}$, which it sets as $\{(\mathbf{r}_i, \mathbf{K}_i^*)\}_{i\in[Q]}$.

3. **Generating the Challenge Ciphertext.** For the challenge ciphertext, do the following.

   (a) Sample $\boldsymbol{\beta}_0, \{\mathbf{u}_j\}_{j\in[\ell]}, \{\mathbf{v}_j\}_{j\in[t]}$ independently and uniformly from $\mathbb{Z}_q^m$.

   (b) Sample $\mathbf{e}_j'' \leftarrow \mathcal{D}_{\mathbb{Z}^m, s_D}$ for $j \in [k]$.

   (c) It computes $\boldsymbol{\beta}_{1j}$ for $j \in [k]$ as

   $$\boldsymbol{\beta}_{1j} = \mathbf{K}_j^\mathsf{T}\, \boldsymbol{\beta}_0 + \mathbf{e}_j'' + \mathbf{b}$$

   (d) It outputs the challenge ciphertext

   $$\mathsf{CT}^* = \Big(\{\mathbf{u}_i\}_{i\in[\ell]}, \{\mathbf{v}_i\}_{i\in[t]}, \mathbf{y}, \boldsymbol{\beta}_0, \{\boldsymbol{\beta}_{1i}\}_{i\in[k]} \Big)$$

See Appendix 8.3 for the analysis of our simulator.

6 Upgrading PHPE Security: Selective to Semi-Adaptive

In this section, we show how to construct a $(1, \mathrm{poly})$-Partially Hiding Predicate Encryption scheme for circuit class $\mathcal{C}$ satisfying semi-adaptive security according to definition 3.1. Our construction, which we denote by SaPH, will make use of two ingredients:

1. A single key[7], $\mathsf{FULL\text{-}SIM}$ secure (see Appendix 8.4 for the definition), functional encryption scheme for the following functionality:

   $$F_{(\mathbf{V}_1,\ldots,\mathbf{V}_k)}(\mathbf{a}_1,\ldots,\mathbf{a}_k) = \sum_{i=1} \mathbf{V}_i \cdot \mathbf{a}_i \mod q$$

   where $\mathbf{V}_i \in \mathbb{Z}_q^{m\times m}$ and $\mathbf{a}_i \in \mathbb{Z}_q^m$ for $i \in [k]$. The parameters k, q, m are input to the setup algorithm. Such a scheme was recently constructed by [ALS16][8]. We will denote this scheme by FuLin.

2. A $(1, \mathrm{poly})$ selectively secure PHPE scheme for the circuit class $\mathcal{C}$, as provided in Section 5. We will denote this scheme by SelPH.

Our approach is quite general and can be used to upgrade the security of any selectively secure functional encryption system where the ciphertext is a linear polynomial with *public* coefficients. This applies to all constructions of functional encryption from the LWE assumption, including [ABB10, AFV11, BGG+14, GVW15].

Our construction is described below.

$\mathsf{SaPH.Setup}(1^\kappa, 1^t, 1^\ell, 1^d)$: Given as input the circuit and the master secret key, do the following:

1. For $i \in [\ell]$, let $(\mathsf{FuLin.PK}_i, \mathsf{FuLin.MSK}_i) \leftarrow \mathsf{FuLin.Setup}\big(1^\kappa, (\mathbb{Z}_q^m)^3\big)$.
2. For $j \in [t]$, let $(\mathsf{FuLin.PK}'_j, \mathsf{FuLin.MSK}'_j) \leftarrow \mathsf{FuLin.Setup}\big(1^\kappa, (\mathbb{Z}_q^m)^3\big)$.

[7] More precisely, we require that the adversary may request the same single function any number of times, but multiple requests for the same function result in the same key.

[8] While the construction in [ALS16] has stateful KeyGen against a general adversary, we only need the single key version which is clearly stateless.

3. Let $(\mathsf{FuLin.PK}_0, \mathsf{FuLin.MSK}_0) \leftarrow \mathsf{FuLin.Setup}\big(1^\kappa, (\mathbb{Z}_q^m)^2\big)$.
4. Let $(\mathsf{FuLin.PK}'_0, \mathsf{FuLin.MSK}'_0) \leftarrow \mathsf{FuLin.Setup}\big(1^\kappa, (\mathbb{Z}_q^m)^3\big)$.
5. Let $\{\mathsf{PRG}\}_{s\in\{0,1\}^\kappa}$ be a family of PRGs with polynomial expansion. Sample a PRG seed, denoted by seed.
6. Output

$$\mathsf{PH.PK} = \big\{\ \mathsf{FuLin.PK}_0,\ \mathsf{FuLin.PK}'_0,\ \{\mathsf{FuLin.PK}_i\}_{i\in[\ell]},\ \{\mathsf{FuLin.PK}'_j\}_{j\in[t]}\ \big\}$$
$$\mathsf{PH.MSK} = \big\{\ \mathsf{seed},\ \ \mathsf{FuLin.MSK}_0,\ \mathsf{FuLin.MSK}'_0,\ \{\mathsf{FuLin.MSK}_i\}_{i\in[\ell]},\ \{\mathsf{FuLin.MSK}'_j\}_{j\in[t]}\ \big\}$$

$\mathsf{SaPH.Enc}\big(\mathsf{PH.PK}, (\mathbf{x}, \mathbf{y}), \mu\big)$: Given as input the master public key, the private attributes $\mathbf{x}$, public attributes $\mathbf{y}$ and message μ, do the following:

1. Sample $\mathbf{s} \leftarrow \mathcal{D}_{\mathbb{Z}^n, s_B}$ and error terms $\mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m, s_B}$ and $\mathbf{e}' \leftarrow \mathcal{D}_{\mathbb{Z}^m, s_D}$.
2. Let $\mathbf{b} = [0, \ldots, 0, \lceil q/2 \rceil \mu]^\mathsf{T} \in \mathbb{Z}_q^m$.
3. Sample $\mathbf{R}_i \leftarrow \{-1, 1\}^{m\times m}$ for $i \in [\ell]$ and $\mathbf{R}'_j \leftarrow \{-1, 1\}^{m\times m}$ for $j \in [t]$.
4. Set

$$\hat{\boldsymbol{\beta}}_0 = \mathsf{FuLin.Enc}\big(\ \mathbf{s}, \mathbf{e}\ \big)^{9}, \qquad \hat{\mathbf{u}}_i = \mathsf{FuLin.Enc}\big(\ \mathbf{s},\ \mathbf{y}[i]\cdot\mathbf{G}^\mathsf{T}\mathbf{s},\ \mathbf{R}_i^\mathsf{T}\mathbf{e}\ \big),$$
$$\hat{\boldsymbol{\beta}}_1 = \mathsf{FuLin.Enc}\big(\ \mathbf{s}, \mathbf{e}', \mathbf{b}\ \big), \qquad \hat{\mathbf{v}}_j = \mathsf{FuLin.Enc}\big(\ \mathbf{s},\ \mathbf{x}[j]\cdot\mathbf{G}^\mathsf{T}\mathbf{s},\ \mathbf{R}'^\mathsf{T}_j\mathbf{e}\ \big)$$

5. Output the ciphertext

$$\mathsf{CT}_\mathbf{y} = \big(\mathbf{y}, \hat{\boldsymbol{\beta}}_0, \hat{\boldsymbol{\beta}}_1, \{\hat{\mathbf{u}}_i\}, \{\hat{\mathbf{v}}_j\}\big)$$

for $i \in [\ell]$, $j \in [t]$.

$\mathsf{SaPH.KeyGen}(\mathsf{PH.MSK}, \widehat{C} \circ \mathsf{IP}_\gamma)$: Given as input the circuit and the master secret key, do the following:

1. Use $\mathsf{PRG}(\mathsf{seed})$ to generate sufficient randomness rand for the $\mathsf{SelPH.Setup}$ algorithm as well as $\{\mathsf{rand}_i\}$, $\{\mathsf{rand}'_j\}$, rand_0, rand'_0 for the $\mathsf{FuLin.KeyGen}$ algorithms.
2. Sample $(\mathsf{SelPH.PK}, \mathsf{SelPH.MSK}) \leftarrow \mathsf{SelPH.Setup}\big(1^\kappa, 1^t, 1^\ell, 1^d, \mathsf{rand}\big)$.
   Parse $\mathsf{SelPH.PK} = \big(\{\mathbf{A}_i\},\ \{\mathbf{B}_j\},\ \mathbf{A},\ \mathbf{P}\ \big)$.
3. Let

$$\mathsf{FuLin.SK}_i \leftarrow \mathsf{FuLin.KeyGen}\big(\ \mathsf{FuLin.MSK}_i, (\mathbf{A}_i^\mathsf{T}, 1, 1),^{10}\ \mathsf{rand}_i\ \big) \qquad \forall i \in [\ell]$$
$$\mathsf{FuLin.SK}'_j \leftarrow \mathsf{FuLin.KeyGen}\big(\ \mathsf{FuLin.MSK}'_j, (\mathbf{B}_j^\mathsf{T}, 1, 1),\ \mathsf{rand}'_j\ \big) \qquad \forall j \in [t]$$
$$\mathsf{FuLin.SK}_0 \leftarrow \mathsf{FuLin.KeyGen}\big(\ \mathsf{FuLin.MSK}_0, (\mathbf{A}^\mathsf{T}, 1),\ \mathsf{rand}_0\ \big)$$
$$\mathsf{FuLin.SK}'_0 \leftarrow \mathsf{FuLin.KeyGen}\big(\ \mathsf{FuLin.MSK}'_0, (\mathbf{P}^\mathsf{T}, 1, 1),\ \mathsf{rand}'_0\ \big)$$

4. Let $\mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\gamma) \leftarrow \mathsf{SelPH.KeyGen}\big(\ \mathsf{SelPH.MSK}, \widehat{C} \circ \mathsf{IP}_\gamma\ \big)$.

[9] Note that we are abusing notation slightly, since the message space of FuLin was set as $\mathbb{Z}_q^m$ but $\mathbf{s} \in \mathbb{Z}_q^n$. However, since $n < m$, we can pad it with zeroes to make it match. We do not explicitly state this for the sake of notational convenience.

[10] Here, 1 is used to denote the $m \times m$ identity matrix.

5. Output

$$\mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\gamma) = \Big(\big(\mathsf{SelPH.PK},\ \mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\gamma)\ \big), \\ \big(\ \{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0\ \big) \Big)$$

$\mathsf{SaPH.Dec}\big(\mathsf{SK}_{\widehat{C} \circ \mathsf{IP}_\gamma}, \mathsf{CT}_{\mathbf{y}}\big)$: Given as input a secret key and a ciphertext, do the following:

1. Let

$$\boldsymbol{\beta}_0 = \mathsf{FuLin.Dec}(\mathsf{FuLin.SK}_0, \hat{\boldsymbol{\beta}}_0), \qquad \mathbf{u}_i = \mathsf{FuLin.Dec}(\mathsf{FuLin.SK}_i, \hat{\mathbf{u}}_i),$$
$$\boldsymbol{\beta}_1 = \mathsf{FuLin.Dec}(\mathsf{FuLin.SK}'_0, \hat{\boldsymbol{\beta}}_1), \qquad \mathbf{v}_j = \mathsf{FuLin.Dec}(\mathsf{FuLin.SK}'_j, \hat{\mathbf{v}}_j)$$

   Let $\mathsf{SelPH.CT} = \big(\ \boldsymbol{\beta}_0,\ \boldsymbol{\beta}_1,\ \{\mathbf{u}_i\},\ \{\mathbf{v}_j\},\ \mathbf{y}\ \big)$.
2. Output $\mu \leftarrow \mathsf{SelPH.Dec}\big(\ \mathsf{SelPH.PK},\ \mathsf{SelPH.CT}, \mathsf{SelPH.SK}\ \big)$.

Correctness. Correctness may be argued using the correctness of FuLin and SelPH.

By correctness of FuLin, the tuple $(\boldsymbol{\beta}_0,\ \boldsymbol{\beta}_1,\ \{\mathbf{u}_i\},\ \{\mathbf{v}_j\})$ produced in the first step of decryption is precisely the ciphertext of the SelPH scheme. More formally, we get:

$$\mathbf{u}_i = \mathsf{FuLin.Dec}(\mathsf{FuLin.SK}_i, \hat{\mathbf{u}}_i) = (\mathbf{A}_i + \mathbf{y}_i \cdot \mathbf{G})^\mathsf{T}\mathbf{s} + \mathbf{R}_i^\mathsf{T}\mathbf{e}$$
$$\mathbf{v}_j = \mathsf{FuLin.Dec}(\mathsf{FuLin.SK}'_j, \hat{\mathbf{v}}_j) = (\mathbf{B}_i + \mathbf{x}_i \cdot \mathbf{G})^\mathsf{T}\mathbf{s} + (\mathbf{R}'_i)^\mathsf{T}\mathbf{e}$$
$$\boldsymbol{\beta}_0 = \mathsf{FuLin.Dec}(\mathsf{FuLin.SK}_0, \hat{\boldsymbol{\beta}}_0) = \mathbf{A}^\mathsf{T}\mathbf{s} + \mathbf{e}$$
$$\boldsymbol{\beta}_1 = \mathsf{FuLin.Dec}(\mathsf{FuLin.SK}'_0, \hat{\boldsymbol{\beta}}_1) = \mathbf{P}^\mathsf{T}\mathbf{s} + \mathbf{e}' + \mathbf{b}$$

Let $\mathsf{SelPH.CT} = \big(\ \boldsymbol{\beta}_0,\ \boldsymbol{\beta}_1,\ \{\mathbf{u}_i\},\ \{\mathbf{v}_j\},\ \mathbf{y}\ \big)$. Then, by correctness of SelPH, the following is correct

$$\mu = \mathsf{SelPH.Dec}\big(\ \mathsf{SelPH.PK},\ \mathsf{SelPH.CT}, \mathsf{SelPH.SK}\ \big)$$

6.1 Security

Theorem 6.1. *Assume that* SelPH *satisfies selective SIM attribute hiding (definition 3.2) and that* FuLin *satisfies* FULL-SIM *security (definition 8.4). Then the scheme* SaPH *satisfies semi-adaptive SIM attribute hiding (definition 3.1).*

Proof. We begin by defining the semi-adaptive simulator SA.Sim. Our simulator will use FuLin.Sim and SelPH.Sim to simulate the view of the semi-adaptive adversary SA.Adv as follows.

Semi Adaptive Simulator $\mathsf{SA.Sim}(1^\kappa)$**:**

1. Invoke $k \in [\ell + t + 2]$ simulators $\mathsf{FuLin.Sim}_k$ to obtain

$$\mathsf{SaPH.PK} = \big(\{\mathsf{FuLin.PK}_i\}, \{\mathsf{FuLin.PK}'_j\}, \mathsf{FuLin.PK}_0, \mathsf{FuLin.PK}'_0\big)$$

   Return this to the adversary SA.Adv.
2. SA.Adv now outputs the PHPE challenge $(\mathbf{x}, \mathbf{y}, \mu, \widehat{C}^* \circ \mathsf{IP}_\gamma)$, upon which the simulator SA.Sim gets $(\mathbf{y}, 1^{|\mathbf{x}|}, \widehat{C}^* \circ \mathsf{IP}_\gamma, \mu)$. Invoke SelPH.Sim to obtain:

$$\big(\ \mathsf{SelPH.PK},\ \mathsf{SelPH.CT},\ \mathsf{SelPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)\ \big) \leftarrow \mathsf{SelPH.Sim}(1^\kappa, \mathbf{y}, 1^{|\mathbf{x}|}, \widehat{C}^* \circ \mathsf{IP}_\gamma, \mu)$$

3. Parse $\mathsf{SelPH.PK} = \big(\{\mathbf{A}_i\},\ \{\mathbf{B}_j\},\ \mathbf{A},\ \mathbf{P} \big)$. Obtain $\ell + t + 2$ function keys for the $\ell + t + 2$ single key FuLin schemes from their respective simulators as:

$$\begin{aligned}
\mathsf{FuLin.SK}_i &\leftarrow \mathsf{FuLin}_i.\mathsf{Sim}(\mathbf{A}_i^\mathsf{T}, 1, 1), \ \ \forall\, i \in [\ell] \\
\mathsf{FuLin.SK}'_j &\leftarrow \mathsf{FuLin}_{\ell+j}.\mathsf{Sim}(\mathbf{B}_j^\mathsf{T}, 1, 1), \ \ \forall\, j \in [t] \\
\mathsf{FuLin.SK}_0 &\leftarrow \mathsf{FuLin.Sim}_{\ell+t+1}(\mathbf{A}^\mathsf{T}, 1) \\
\mathsf{FuLin.SK}'_0 &\leftarrow \mathsf{FuLin.Sim}_{\ell+t+2}(\mathbf{P}^\mathsf{T}, 1, 1)
\end{aligned}$$

Return the 1-key $\mathsf{SaPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)$ to SA.Adv as:

$$\Big(\mathsf{SelPH.PK},\ \mathsf{SelPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma),\ \{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0\ \Big)$$

4. Parse

$$\mathsf{SelPH.CT} = (\ \mathbf{y},\ \{\mathbf{u}_i\},\ \{\mathbf{v}_j\},\ \boldsymbol{\beta}_0,\ \boldsymbol{\beta}_1)$$

$$\begin{aligned}
\text{Let} \ \ \hat{\mathbf{u}}_i &\leftarrow \mathsf{FuLin}_i.\mathsf{Sim}\big(\ (\mathbf{A}_i^\mathsf{T}, 1, 1),\ \mathbf{u}_i\big) \ \ \forall i \in [\ell] \\
\hat{\mathbf{v}}_j &\leftarrow \mathsf{FuLin}_{\ell+j}.\mathsf{Sim}\big(\ (\mathbf{B}_j^\mathsf{T}, 1, 1),\ \mathbf{v}_j\big) \ \ \forall j \in [t] \\
\hat{\boldsymbol{\beta}}_0 &\leftarrow \mathsf{FuLin.Sim}_{\ell+t+1}\big((\mathbf{A}^\mathsf{T}, 1), \boldsymbol{\beta}_0\big) \\
\hat{\boldsymbol{\beta}}_1 &\leftarrow \mathsf{FuLin.Sim}_{\ell+t+2}\big((\mathbf{P}^\mathsf{T}, 1, 1), \boldsymbol{\beta}_1\big)
\end{aligned}$$

Return the challenge ciphertext to SA.Adv as:

$$\mathsf{SaPH.CT} = \big(\mathbf{y},\ \{\hat{\mathbf{u}}_i\},\ \{\hat{\mathbf{v}}_j\},\ \hat{\boldsymbol{\beta}}_0,\ \hat{\boldsymbol{\beta}}_1\ \big)$$

5. Adversary may request 0-keys $\widehat{C} \circ \mathsf{IP}_\rho$, which are forwarded to SelPH.Sim to obtain $\mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\rho)$. Then, the key $\mathsf{SaPH.SK}(\widehat{C} \circ \mathsf{IP}_\rho)$ is defined as:

$$\Big(\mathsf{SelPH.PK},\ \mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\rho),\ \{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0\ \Big)$$

and returned to the adversary SA.Adv.

Analysis of SA.Sim: Correctness of the simulator SA.Sim follows from correctness of FuLin.Sim and SelPH.Sim. We will argue that the real world is indistinguishable from the simulated world via a sequence of hybrids, which are defined below.

Hybrid 0: Ideal world SaPH.

Hybrid 1: In this hybrid, replace the simulator SelPH.Sim by the algorithms SelPH.Setup, SelPH.KeyGen used by SaPH.KeyGen.

Hybrid 2: In this hybrid, replace the simulator FuLin.Sim by the algorithms FuLin.Setup, FuLin.Enc and FuLin.KeyGen. This is the real world.

We now argue that consecutive hybrids are indistinguishable.

Lemma 6.2. *Hybrids 0 and 1 are computationally indistinguishable by the selective security of the* SelPH *scheme.*

Proof. Assume there exists an adversary $\mathcal{A}$ who distinguishes between Hybrids 0 and 1. We will construct an adversary $\mathcal{B}$ to break selective security of the SelPH scheme. The adversary $\mathcal{B}$ does the following:

1. $\mathcal{B}$ runs simulators $\mathsf{FuLin}_k.\mathsf{Sim}$ to obtain $\mathsf{FuLin}_k.\mathsf{PK}$ for $k \in [\ell + t + 2]$. It sets $\mathsf{SaPH.PK} = \{\mathsf{FuLin}_k.\mathsf{PK}\}$ and provides this to $\mathcal{A}$.

2. $\mathcal{A}$ returns the challenge $(\mathbf{x}, \mathbf{y}, \mu, \widehat{C}^* \circ \mathsf{IP}_\gamma)$, which $\mathcal{B}$ outputs as its challenge.

3. $\mathcal{B}$ is provided $\big(\ \mathsf{SelPH.PK},\ \mathsf{SelPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma),\ \mathsf{SelPH.CT}_{\mathbf{y}}\ \big)$. It does the following:

   (a) To construct $\mathsf{SaPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)$, it does the following.

      i. It requests FuLin keys from the simulators $\mathsf{FuLin}_k.\mathsf{Sim}$ for $k \in [\ell + t + 2]$ as:

$$\{\mathsf{FuLin.SK}_i \text{ for function } (\mathbf{A}_i^\mathsf{T}, 1, 1)\},\ \mathsf{FuLin.SK}_0 \text{ for function } (\mathbf{A}^\mathsf{T}, 1)$$
$$\{\mathsf{FuLin.SK}'_j \text{ for function } (\mathbf{B}_j^\mathsf{T}, 1, 1)\},\ \mathsf{FuLin.SK}'_0 \text{ for function } (\mathbf{P}^\mathsf{T}, 1, 1)$$

      ii. It sets as $\mathsf{SaPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)$ the tuple:

$$\Big(\mathsf{SelPH.PK},\ \mathsf{SelPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma),\ \{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0\ \Big)$$

   (b) To construct $\mathsf{SaPH.CT}$, it does the following:

      i. Parse $\mathsf{SelPH.CT}_{\mathbf{y}} = (\ \mathbf{y},\ \{\mathbf{u}_i\},\ \{\mathbf{v}_j\},\ \boldsymbol{\beta}_0,\ \boldsymbol{\beta}_1)$.

      ii. Invoke the simulators $\mathsf{FuLin}_k.\mathsf{Sim}$ for $k \in [\ell + t + 2]$ as:

$$\begin{aligned}
\text{Let } \ \hat{\mathbf{u}}_i &\leftarrow \mathsf{FuLin}_i.\mathsf{Sim}\big(\ (\mathbf{A}_i^\mathsf{T}, 1, 1),\ \mathbf{u}_i\big)\ \ \forall i \in [\ell] \\
\hat{\mathbf{v}}_j &\leftarrow \mathsf{FuLin}_{\ell+j}.\mathsf{Sim}\big(\ (\mathbf{B}_j^\mathsf{T}, 1, 1),\ \mathbf{v}_j\big)\ \ \forall j \in [t] \\
\hat{\boldsymbol{\beta}}_0 &\leftarrow \mathsf{FuLin.Sim}_{\ell+t+1}\big((\mathbf{A}^\mathsf{T}, 1), \boldsymbol{\beta}_0\big) \\
\hat{\boldsymbol{\beta}}_1 &\leftarrow \mathsf{FuLin.Sim}_{\ell+t+2}\big((\mathbf{P}^\mathsf{T}, 1, 1), \boldsymbol{\beta}_1\big)
\end{aligned}$$

      iii. Return the challenge ciphertext to $\mathcal{A}$ as:

$$\mathsf{SaPH.CT} = \big(\mathbf{y},\ \{\hat{\mathbf{u}}_i\},\ \{\hat{\mathbf{v}}_j\},\ \hat{\boldsymbol{\beta}}_0,\ \hat{\boldsymbol{\beta}}_1\ \big)$$

4. If $\mathcal{A}$ makes a query $\widehat{C} \circ \mathsf{IP}_\rho$, invoke $\mathsf{SelPH.Sim}(\widehat{C} \circ \mathsf{IP}_\rho)$ to obtain $\mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\rho)$. Return $\mathsf{SaPH.SK}(\widehat{C} \circ \mathsf{IP}_\rho)$ as

$$\Big(\mathsf{SelPH.PK},\ \mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\rho),\ \{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0\ \Big)$$

   where $\mathsf{SelPH.PK}$ and $\{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0$ are as generated in the previous step.

5. Finally, when $\mathcal{A}$ outputs some α, output the same.

When the SelPH objects come from the simulator, then $\mathcal{A}$ sees the distribution of Hybrid 0, and when they come from the real scheme, $\mathcal{A}$ sees the distribution of Hybrid 1. Thus, if it can distinguish between these, then $\mathcal{B}$ is a successful adversary against the SelPH scheme. □

Lemma 6.3. *Hybrids 1 and 2 are indistinguishable by the* FULL-SIM *security of the* FuLin *scheme.*

Proof. Assume there exists adversary $\mathcal{A}$ who distinguishes between Hybrids 1 and 2. We will construct an adversary $\mathcal{B}$ to break FULL-SIM security of the FuLin scheme.

1. $\mathcal{B}$ receives the $\mathsf{FuLin}_k.\mathsf{PK}$ for $k \in [\ell + t + 2]$. It provides this set to $\mathcal{A}$ as the SaPH.PK.
2. $\mathcal{A}$ returns the challenge $(\mathbf{x}, \mathbf{y}, \mu, \widehat{C}^* \circ \mathsf{IP}_\gamma)$. $\mathcal{B}$ does the following:
   (a) To construct $\mathsf{SaPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)$:
      i. It computes SelPH.PK and $\mathsf{SelPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)$ by itself using the SelPH algorithms.
      ii. Now, $\mathsf{SelPH.PK} = \big(\{\mathbf{A}_i\},\ \{\mathbf{B}_j\},\ \mathbf{A},\ \mathbf{P}\ \big)$. It requests $\ell + t + 2$ FuLin keys:
      $$\{\mathsf{FuLin.SK}_i \text{ for function } (\mathbf{A}_i^\mathsf{T}, 1, 1)\},\ \mathsf{FuLin.SK}_0 \text{ for function } (\mathbf{A}^\mathsf{T}, 1)$$
      $$\{\mathsf{FuLin.SK}'_j \text{ for function } (\mathbf{B}_j^\mathsf{T}, 1, 1)\},\ \mathsf{FuLin.SK}'_0 \text{ for function } (\mathbf{P}^\mathsf{T}, 1, 1)$$
      iii. It sets as $\mathsf{SaPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)$ the tuple:
      $$\Big(\mathsf{SelPH.PK},\ \mathsf{SelPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma),\ \{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0\ \Big)$$
   (b) To construct SaPH.CT: It chooses $\ell + t + 2$ challenge messages (sampled as in SelPH.Enc), one for each instance of FuLin, and outputs:
   $$\big(\ \mathbf{s},\ \mathbf{y}[i] \cdot \mathbf{G}^\mathsf{T}\mathbf{s},\ \mathbf{R}_i^\mathsf{T}\mathbf{e}\ \big)_{i \in [\ell]}$$
   $$\big(\ \mathbf{s},\ \mathbf{x}[j] \cdot \mathbf{G}^\mathsf{T}\mathbf{s},\ \mathbf{R}'^\mathsf{T}_j\mathbf{e}\ \big)_{j \in [t]}$$
   $$\big(\ \mathbf{s}, \mathbf{e}\ \big), \quad \big(\ \mathbf{s}, \mathbf{e}', \mathbf{b}\ \big)$$
   It receives $\big(\ \{\hat{\mathbf{u}}_i\},\ \{\hat{\mathbf{v}}_j\},\ \hat{\boldsymbol{\beta}}_0,\ \hat{\boldsymbol{\beta}}_1\ \big)$ which it sets as SaPH.CT.
   (c) $\mathcal{B}$ outputs SaPH.CT and $\mathsf{SaPH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)$.
3. $\mathcal{A}$ may request a 0-query $\widehat{C} \circ \mathsf{IP}_\rho$. It computes $\mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\rho)$ itself using the SelPH.KeyGen algorithm. It outputs
   $$\Big(\mathsf{SelPH.PK},\ \mathsf{SelPH.SK}(\widehat{C} \circ \mathsf{IP}_\rho),\ \{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0\ \Big)$$
   where SelPH.PK and $\{\mathsf{FuLin.SK}_i\},\ \{\mathsf{FuLin.SK}'_j\},\ \mathsf{FuLin.SK}_0,\ \mathsf{FuLin.SK}'_0$ are as generated in the previous step.
4. When $\mathcal{A}$ outputs some state α, $\mathcal{B}$ outputs the same.

When the FuLin objects come from the simulator, then $\mathcal{A}$ sees the distribution of Hybrid 1, and when they come from the real scheme, $\mathcal{A}$ sees the distribution of Hybrid 2. Thus, if it can distinguish between these, then $\mathcal{B}$ is a successful adversary against the FuLin scheme. □

Thus, we have proved that the simulator is correct, establishing the theorem. □

7 $(1, \mathrm{poly})$-Functional Encryption.

In this section, we construct our $(1, \mathrm{poly})$-functional encryption scheme. The ciphertext of the construction is succinct, providing a unification of the results [GKP+13, GVW15]. Our construction of $(1, \mathrm{poly})$-functional encryption uses $(1, \mathrm{poly})$-partially hiding predicate encryption and fully homomorphic encryption in a manner similar to [GVW15, Section 4].

7.1 Construction

We begin with an overview of the main ideas in the construction. Let us recall the $(0, \text{poly})$-FE scheme constructed by [GVW15]. The scheme makes use of two ingredients, namely, a $(0, \text{poly})$-PHPE scheme for circuits, and a fully homomorphic encryption scheme for circuits. The ciphertext of $(0, \text{poly})$-FE corresponding to an attribute $\mathbf{a}$ is a PHPE ciphertext corresponding to $(\widehat{\mathbf{a}}, \mathbf{t})$ where $\widehat{\mathbf{a}}$ is the FHE encryption of $\mathbf{a}$, and corresponds to the public attributes in PHPE, while $\mathbf{t}$ is the FHE secret key and corresponds to the private attributes in PHPE.

The secret key corresponding to circuit C in the $(0, \text{poly})$-FE scheme is a set of PHPE secret keys $\{ \widehat{C} \circ \mathsf{IP}_\gamma \}_{\gamma \in [\lfloor p/2 \rfloor - B, \lfloor p/2 \rfloor + B]}$ where:

$$\begin{aligned} \widehat{C} \circ \mathsf{IP}_\gamma(\mathbf{x}, \mathbf{y}) = 1 \quad \text{if } \langle \mathbf{x}, \widehat{C}(\mathbf{y}) \rangle = \gamma \\ = 0 \text{ otherwise.} \end{aligned}$$

The decryptor executes the homomorphic ciphertext evaluation procedure for circuit $\mathsf{FHE.Eval}(\cdot, C)$ on the attributes $\widehat{\mathbf{a}}$ embedded in the PHPE ciphertext as in [BGG+14] to obtain a ciphertext corresponding to public attributes $\widehat{C(\mathbf{a})}$, where $\widehat{C(\mathbf{a})}$ is an FHE encryption of $C(\mathbf{a})$. Now, when $C(\mathbf{a}) = 1$, then by correctness of FHE, there exists a noise term $\gamma \in [\lfloor p/2 \rfloor - B, \lfloor p/2 \rfloor + B]$ such that $\langle \mathbf{t}, \widehat{C(\mathbf{a})} \rangle = \gamma$. The decryptor tries keys corresponding to all possible γ within the aforementioned range to ascertain whether $\widehat{C} \circ \mathsf{IP}_\gamma(\widehat{\mathbf{a}}, \mathbf{t}) = 1$. Note that this step makes it crucial that the FHE decryption range be polynomial in size. Fortunately, as noted by [GVW15], this can be ensured by the modulus reduction technique in FHE schemes [BGV12, GSW13, BV14], which allows a superpolynomial modulus to be scaled down to polynomial size.

The first idea in building $(1, \text{poly})$-FE is to replace the use of $(0, \text{poly})$-PHPE in the above transformation by the $(1, \text{poly})$ PHPE constructed in Section 5. However, as discussed in Attack #3, Section 4, such a straightforward adaptation leads to vulnerabilities. This is because decryption using a 1-key allows the decryptor to learn the exact inner product of the FHE ciphertext $\widehat{C(\mathbf{a})}$ and the FHE secret key $\mathbf{t}$ rather than the threshold inner product corresponding to FHE decryption. This lets her obtain leakage on the noise terms used to construct $\widehat{\mathbf{a}}$, which is problematic. We will denote the noise used in the construction of the FHE ciphertext $\widehat{\mathbf{a}}$ by $\mathsf{Noise}(\widehat{\mathbf{a}})$.

Overcoming Leakage on FHE noise. For a single 1-key, there is a natural way out, via "noise flooding" or "noise smudging" [Gen09, GKPV10, AJLA+12]. To prevent leakage on $\mathsf{Noise}(\widehat{\mathbf{a}})$, we may augment the FHE evaluation circuit with a "flooding" operation, which, after computing $\mathsf{FHE.Eval}(\widehat{\mathbf{a}}, C)$ adds to it an encryption of 0 with large noise η to drown out the effects of $\mathsf{Noise}(\widehat{\mathbf{a}})$. This idea is complicated by the fact that our construction of $(1, \text{poly})$-FE must use an FHE scheme whose final modulus is polynomial in size, whereas η must be chosen to satisfy:

$$\mathsf{Noise}(\mathsf{FHE.Eval}(\widehat{\mathbf{a}}, C)) + \eta \overset{s}{\approx} \eta \tag{7.1}$$

so that, by Lemma 2.2, it drowns the effects of $\mathsf{Noise}(\widehat{\mathbf{a}})$. The above constraint may necessitate η, and hence the FHE modulus, to be superpolynomial in the security parameter.

Fortunately, we can work around this difficulty by performing FHE modulus reduction after flooding. Then, η can be superpolynomial in the security parameter to obliterate the dependency of the revealed noise on the initial noise, while letting the final FHE modulus still be polynomial. Another method is to use the "sanitization" operation [DS16], which will result in better parameters for this step – however, since it does not improve our overall parameters, we do not discuss this.

Formally, we require a PHPE scheme for the circuit family $\mathcal{C}_{\mathsf{PHPE}}$ where $\widehat{C} \circ \mathsf{IP} \in \mathcal{C}_{\mathsf{PHPE}}$ is defined as follows. Let the private attributes $\mathbf{x} = \mathbf{t}$ where $\mathbf{t}$ is the FHE secret, and public attributes $\mathbf{y} = (\widehat{\mathbf{a}}, \widehat{0})$, where $\widehat{0}$ is an FHE encryption of the bit 0, with large noise η. Then, define:

$$\widehat{C}(\widehat{0}, \widehat{\mathbf{a}}) = \mathsf{FHE.Scale}_{q,p}\big(\ \mathsf{FHE.Eval}(\widehat{\mathbf{a}}, C) + \widehat{0}\big)$$
$$\widehat{C} \circ \mathsf{IP}\big(\ \mathbf{t}, \widehat{0}, \widehat{\mathbf{a}}\ \big) = \Big\langle \mathbf{t},\ \widehat{C}(\widehat{0}, \widehat{\mathbf{a}}) \Big\rangle \mod p$$
$$\widehat{C} \circ \mathsf{IP}_\gamma\big(\ \mathbf{t}, \widehat{0}, \widehat{\mathbf{a}}\ \big) = 1 \ \text{ iff } \ \widehat{C} \circ \mathsf{IP}\big(\ \mathbf{t}, \widehat{0}, \widehat{\mathbf{a}}\ \big) = \gamma,\ 0 \text{ otherwise.}$$

Above, $\mathsf{FHE.Eval}$ is the FHE ciphertext evaluation algorithm, and $\mathsf{FHE.Scale}$ is the modulus reduction algorithm described in Section 3.3. Recall that $\mathsf{FHE.Scale}_{q,p}$ takes as input an FHE ciphertext that lives modulo q and reduces it to a ciphertext that lives modulo p. For the sake of brevity, we abuse notation and do not explicitly include the inputs (q, p) in the inputs to $\widehat{C} \circ \mathsf{IP}$.

Construction. We now proceed to describe the construction.

$\mathsf{FE.Setup}(1^\kappa, 1^k, 1^d)$: The setup algorithm takes the security parameter κ, the attribute length k and the function depth d and does the following:

1. Choose the FHE modulus q in which $\mathsf{FHE.Eval}(\cdot, \cdot)$ will be computed and the FHE modulus $p \in \mathrm{poly}(\kappa)$ in which decryption will be performed as per Section 3.3.
2. Invoke the setup algorithm for the PHPE scheme for family $\mathcal{C}_{\mathsf{PHPE}}$ to get:

$$(\mathsf{PH.PK}, \mathsf{PH.MSK}) \leftarrow \mathsf{PH.Setup}(1^\kappa, 1^t, 1^\ell, 1^{d'})$$

   where length of private attributes $t = |\mathsf{FHE.SK}|$, length of public attributes ℓ is the length of an FHE encryption of $k+1$ bits corresponding to the attributes $\mathbf{a}$ and 0, i.e. $\ell = (k+1) \cdot |\mathsf{FHE.CT}|$ and d' is the bound on the augmented FHE evaluation circuit.
3. Output $(\mathsf{PK} = \mathsf{PH.PK},\ \mathsf{MSK} = \mathsf{PH.MSK})$.

$\mathsf{FE.Keygen}(\mathsf{MSK}, C)$: The key generation algorithm takes as input the master secret key MSK and a circuit C. It does the following:

1. Let $R \triangleq [\lfloor p/2 \rfloor - B, \lfloor p/2 \rfloor + B]$. Compute the circuit $\widehat{C} \circ \mathsf{IP}_\gamma$ as described above for $\gamma \in R$.
2. For $\gamma \in R$, compute

$$\mathsf{PH.SK}_{\widehat{C} \circ \mathsf{IP}_\gamma} \leftarrow \mathsf{PH.KeyGen}(\mathsf{PH.MSK}, \widehat{C} \circ \mathsf{IP}_\gamma)$$

3. Output the secret key as $\mathsf{SK}_C = \{\mathsf{PH.SK}_{\widehat{C} \circ \mathsf{IP}_\gamma}\}_{\gamma \in R}$.

$\mathsf{FE.Enc}(\mathsf{PK}, \mathbf{a}, \mu)$: The encryption algorithm does the following:

1. Sample a fresh FHE secret key $\mathsf{FHE.SK}$, and denote it by $\mathbf{t}$.
2. Compute an FHE encryption of $\mathbf{a}$ to get $\widehat{\mathbf{a}} = \mathsf{FHE.Enc}(\ \mathbf{t}, \mathbf{a}\ )$.
3. Sample η to satisfy equation 7.1 and compute an FHE encryption of 0 with noise η as $\widehat{0}$.
4. Set public attributes $\mathbf{y} = (\widehat{\mathbf{a}}, \widehat{0})$ and private attributes $\mathbf{x} = \mathbf{t}$.
5. Compute $\mathsf{PH.CT}_{\widehat{\mathbf{a}}, \widehat{0}} = \mathsf{PH.Enc}\big(\ \mathsf{PH.PK}, (\mathbf{x}, \mathbf{y}), \mu\big)$.

6. Output $\mathsf{CT}_{\mathbf{a}} = \left(\widehat{\mathbf{a}}, \widehat{0}, \mathsf{PH.CT}_{\widehat{\mathbf{a}},\widehat{0}} \right)$.

$\mathsf{FE.Dec}(\mathsf{SK}_C, \mathsf{CT}_{\mathbf{a}})$: Do the following:

1. Parse SK_C as the set $\{\mathsf{PH.SK}_{\widehat{C}\circ\mathsf{IP}_\gamma}\}_{\gamma\in R}$.
2. For $\gamma \in R$, let $\tau_\gamma = \mathsf{PH.Dec}\left(\mathsf{CT}_{\mathbf{a}}, \mathsf{PH.SK}_{\widehat{C}\circ\mathsf{IP}_\gamma} \right)$. If there exists some value γ' for which $\tau_{\gamma'} \neq \perp$, then output $\mu = \tau_{\gamma'}$, else output $\perp$.

Correctness. Correctness follows from correctness of PHPE. Recall that

$$\widehat{C} \circ \mathsf{IP}_\gamma(\mathbf{t}, \widehat{0}, \widehat{\mathbf{a}}) = 1 \quad \text{if } \langle \mathbf{t}, \widehat{C}(\widehat{0},\ \widehat{\mathbf{a}}) \rangle = \gamma, \quad \text{and} \quad 0 \text{ otherwise.}$$

Note that $\widehat{C}(\widehat{0}, \widehat{\mathbf{a}}) = \mathsf{FHE.Scale}\left(\mathsf{FHE.Eval}(C, \widehat{\mathbf{a}}) + \widehat{0},\ q, p \right)$ is a modulo p FHE ciphertext of message $C(\mathbf{a})$ by properties of FHE (see Section 3.3). Now, if $C(\mathbf{a}) = 1$, then for some $\rho \in [\lfloor p/2 \rfloor - B, \lfloor p/2 \rfloor + B]$, we have that:

$$\left\langle \mathbf{t},\ \widehat{C}(\widehat{0}, \widehat{\mathbf{a}}) \right\rangle = \rho \mod p$$

Hence $\mathsf{PH.Dec}\left(\mathsf{SK}_{\widehat{C}\circ\mathsf{IP}_\gamma},\ \widehat{C} \circ \mathsf{IP}_\gamma,\ \mathsf{CT}_{\widehat{\mathbf{a}},\widehat{0}},\ \widehat{\mathbf{a}}, \widehat{0}\right) = \mu$ iff $\gamma = \rho$. On the other hand, if $C(\mathbf{a}) = 0$, then ρ must lie outside $[\lfloor p/2 \rfloor - B, \lfloor p/2 \rfloor + B]$. Hence $\mathsf{PH.Dec}$ returns $\perp$ by correctness of PHPE.

7.2 Proof of Security

Next, we argue that the above scheme satisfies semi-adaptive security.

Theorem 7.1. *The* $(1, \text{poly})$ *functional encryption scheme described above is secure according to definition 3.5.*

Proof. We construct a simulator $\mathsf{FE.Sim}$ as required by definition 3.5 as follows.

Simulator $\mathsf{FE.Sim}(1^\kappa)$**.** The simulator is described as follows.

1. It invokes $\mathsf{PHPE.Sim}(1^\kappa)$ to obtain the public parameters and returns these.
2. The FE adversary outputs $(\mathbf{a}, \mu, C^*)$ upon which, $\mathsf{FE.Sim}$ obtains $(1^{|\mathbf{a}|}, \mu, C^*)$. It does the following:
   (a) It samples an FHE secret key $\mathsf{FHE.SK}$ and sets $\widehat{\mathbf{a}} = \mathsf{FHE.Enc}(\mathsf{FHE.SK}, \mathbf{0})$ and $\widehat{0} = \mathsf{FHE.Enc}(\mathsf{FHE.SK}, 0)$.
   (b) It samples γ_q to satisfy Equation 7.1. Let γ denote its scaled down version modulo p. It computes $\widehat{C}^* \circ \mathsf{IP}_\gamma$ as described above.
   (c) It invokes $\mathsf{PHPE.Sim}\left(\widehat{\mathbf{a}}, \widehat{0},\ 1^{|\mathsf{FHE.SK}|},\ \widehat{C}^* \circ \mathsf{IP}_\gamma,\ \mu \right)$ to obtain $\left(\mathsf{PH.CT},\ \mathsf{PH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma) \right)$.
   (d) For $\rho \in R \setminus \gamma$, it constructs $\widehat{C}^* \circ \mathsf{IP}_\rho$ and sends these queries to $\mathsf{PHPE.Sim}$. It receives $\mathsf{PH.SK}(\widehat{C}^* \circ \mathsf{IP}_\rho)$.
   (e) It outputs $\left(\mathsf{PH.CT},\ \{\mathsf{PH.SK}(\widehat{C}^* \circ \mathsf{IP}_\rho)\}_{\rho\in\mathsf{R}} \right)$.
3. When Adv makes any query C, $\mathsf{FE.Sim}$ transforms it into $\{\widehat{C} \circ \mathsf{IP}_\rho\}_{\rho\in\mathsf{R}}$ and sends this to $\mathsf{PHPE.Sim}$. It returns the set of received keys to Adv. Note that these are 0-keys.
4. When Adv outputs α, output the same.

We argue that the simulator is correct via a series of hybrids.

Hybrid 0: The real experiment.

Hybrid 1: In this hybrid, the FHE encryption $\widehat{\mathbf{a}}$ is generated honestly but the remainder of the experiment is simulated.

Hybrid 2: This is the simulated experiment.

We now argue that consecutive hybrids are indistinguishable.

Lemma 7.2. *Hybrids 0 and 1 are computationally indistinguishable assuming security of* $(1, \mathrm{poly})$ PHPE.

Proof. Assume there exists an adversary Adv who distinguishes between the two hybrids. We use these to construct Adv' that breaks the security of $(1, \mathrm{poly})$ PHPE. Adv' behaves as follows:

1. Obtain the public key $\mathsf{PHPE.PK}$ and output this to Adv.
2. Adv outputs the tuple $(\mathbf{a}, \mu, C^*)$ such that $C^*(\mathbf{a}) = 1$. Do the following.
   (a) Sample an FHE secret key $\mathsf{FHE.SK}$ and set $\widehat{\mathbf{a}} = \mathsf{FHE.Enc}(\mathsf{FHE.SK}, \mathbf{a})$.
   (b) Sample γ_q to satisfy Equation 7.1 and let γ be its scaled down version modulo p. Compute $\widehat{C}^* \circ \mathsf{IP}_\gamma$ as described above.
   (c) Compute η so that $\mathsf{Noise}\big(\mathsf{FHE.Eval}(\widehat{\mathbf{a}}, C^*)\big) + \eta = \gamma_q$.
   (d) Output $\big(\ \mathsf{FHE.SK},\ (\widehat{\mathbf{a}}, \widehat{0}),\ \mu,\ \widehat{C}^* \circ \mathsf{IP}_\gamma\ \big)$ as the PHPE. challenge.
   Obtain $\big(\ \mathsf{PH.CT}_{\widehat{\mathbf{a}},\widehat{0}},\ \mathsf{PH.SK}(\widehat{C}^* \circ \mathsf{IP}_\gamma)\ \big)$.
   (e) Output queries $\widehat{C}^* \circ \mathsf{IP}_\rho$ for $\rho \in R \setminus \gamma$ to receive $\mathsf{PH.SK}(\widehat{C}^* \circ \mathsf{IP}_\rho)$.
   (f) Output $\mathsf{FE.CT} = (\widehat{\mathbf{a}}, \widehat{0}\ \mathsf{PH.CT}_{\widehat{\mathbf{a}},\widehat{0}})$ and $\mathsf{FE.SK} = \{\mathsf{PH.SK}(\widehat{C}^* \circ \mathsf{IP}_\rho)\}_{\rho \in R}$.
3. When Adv makes any query C, transform it into $\{\widehat{C} \circ \mathsf{IP}_\rho\}_{\rho \in \mathsf{R}}$ and send this to $\mathsf{PHPE.Sim}$. Return the set of received keys to Adv.
4. When Adv outputs α, output the same.

Note that the only difference between the two distributions is that in Hybrid 0, the algorithms $\mathsf{PH.Setup}$, $\mathsf{PH.KeyGen}$ and $\mathsf{PH.Enc}$ are used, whereas in Hybrid 1, the simulator $\mathsf{PH.Sim}$ is used. Thus, if there exists a distinguisher that can distinguish between the two hybrids, it is immediate that the real and simulated worlds in the PHPE experiment can be distinguished. □

Lemma 7.3. *Hybrids* 1 *and* 2 *are indistinguishable assuming semantic security of FHE.*

Proof. The only difference between Hybrids 1 and 2 is that in the former, the FHE ciphertext is generated honestly as $\widehat{\mathbf{a}}$ whereas in the latter, the FHE ciphertext is an encryption of $\mathbf{0}$. Hence, given an adversary who can distinguish beween these two hybrids, we can construct an adversary who breaks the semantic security of FHE. For details, we refer the reader to [GVW15, Lemma 4.4]. □

□

7.3 Generalizing to handle Q queries.

In this section, we discuss how to generalize our $(1, \text{poly})$-FE scheme to (Q, poly)-FE for any polynomial Q fixed in advance. We will follow the template for $(1, \text{poly})$-FE but replace the use of $(1, \text{poly})$-PHPE with (Q, poly)-PHPE constructed in Section 5.3. It remains to perform the flooding operation for Q keys instead of 1, in a manner that allows KeyGen to remain stateless. However, this can easily be handled, by using tricks similar to Section 5.3.

In more detail, the encryptor now creates k FHE encryptions of 0 with noise terms $\eta_1, \ldots, \eta_k$ (for some k to be chosen) instead of a single one, sets public attributes $\mathbf{y} = (\widehat{\mathbf{a}}, \widehat{0}_1, \ldots, \widehat{0}_k)$and private attributes $\mathbf{x} = \mathbf{t}$ as before. It outputs $\mathsf{PH.CT} = \mathsf{PH.Enc}\big(\mathsf{PH.PK}, (\mathbf{y}, \mathbf{x}), \mu\big)$.

The key generator chooses a random binary vector $\mathbf{r} \leftarrow \{0,1\}^k$ with hamming weight v, and computes the PHPE secret key for $\widehat{C} \circ \mathsf{IP}_\gamma$ for $\gamma \in [\lfloor p/2 \rfloor - B, \lfloor p/2 \rfloor + B]$ where $\widehat{C} \circ \mathsf{IP}_\gamma$ is slightly modified so that in place of adding $\widehat{0}$ directly to the requisite ciphertext, it adds with $\sum_{j \in [k]} r_j \widehat{0}_j$. By suitable choice of the parameters (k, v) as in Section 5.3, for Q queries, the vectors $\mathbf{r}_1, \ldots, \mathbf{r}_Q$ chosen as above represent a collection of cover free sets. This implies that each of the Q noise terms revealed to the decryptor, corresponding to decryption performed using each of the Q 1-keys, contains a fresh and unique η_j noise term which does not appear in any other decryption equation. This suffices for flooding the problematic noise terms $\mathsf{Noise}(\widehat{\mathbf{a}})$ as in the case of $(1, \text{poly})$-FE. We defer details to the full version of the paper.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *Asiacrypt*, 2011.

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Advances in Cryptology – EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, 2012.

[Ajt99] Miklos Ajtai. Generating hard instances of the short basis problem. In *ICALP*, volume 1644 of *LNCS*, pages 1–9. Springer, 1999.

[ALS16] Shweta Agrawal, Benoit Libert, and Damien Stehle. Fully secure functional encryption for linear functions from standard assumptions. In *CRYPTO*, 2016.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 297–314, 2014.

[AR16] Shweta Agrawal and Alon Rosen. Online-offline functional encryption for bounded collusions. Eprint/2016, 2016.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, 2014.

[BV16] Zvika Brakerski and Vinod Vaikuntanathan. Circuit-abe from lwe: Unbounded attributes and semi-adaptive security. Cryptology ePrint Archive, Report 2016/118, 2016. http://eprint.iacr.org/2016/118.

[BW06] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, pages 290–307, 2006.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[CFL+16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new clt multilinear map over the integers. In *Eurocrypt*, 2016.

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrede Lepoint, Hemanta K Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New mmap attacks and their limitations. In *Advances in Cryptology–CRYPTO 2015*, pages 247–266. Springer, 2015.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.

[CHL+15] J.-H. Cheon, K. Han, C. Lee, H. Ryu, and D. Stehlé. Cryptanalysis of the multilinear map over the integers. In *Proc. of EUROCRYPT*, volume 9056 of *LNCS*, pages 3–12. Springer, 2015.

[CJL] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low level encoding of zero. Eprint 2016/139.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 476–493, 2013.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[CW14] Jie Chen and Hoeteck Wee. Semi-adaptive attribute-based encryption and improved delegation for boolean formula. In *Security and Cryptography for Networks: 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, 2014.

[DS16] Léo Ducas and Damien Stehlé. Sanitization of fhe ciphertexts. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–310. Springer, 2016.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013. http://eprint.iacr.org/.

[GGH+13c] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 498–527, 2015.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. In *IACR Cryptology ePrint Archive*, volume 2014, page 666, 2014.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *ITCS*, 2010.

[GKW16] Rishab Goyal, Venkata Koppula, and Brent Waters. Semi-adaptive security and bundling functionalities made generic and easy. Cryptology ePrint Archive, Report 2016/317, 2016. http://eprint.iacr.org/2016/317.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[GTKP+13] S. Goldwasser, Y. Tauman Kalai, R. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proc. of STOC*, pages 555–564. ACM Press, 2013.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions from multiparty computation. In *CRYPTO*, 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute based encryption for circuits. In *STOC*, 2013.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *Crypto*, 2015.

[HJ15] Y. Hu and H. Jia. Cryptanalysis of GGH map. Cryptology ePrint Archive: Report 2015/301, 2015.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 700–718, 2012.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over ggh13. In *Crypto*, 2016.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

[Pei13] Chris Peikert. Lattices....to cryptography. web.eecs.umich.edu/~cpeikert/pubs/slides-visions.pdf, 2013.

[PR12] Omkant Pandey and Yannis Rouselakis. Property preserving symmetric encryption. In *Advances in Cryptology – EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, 2012.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J.ACM*, 56(6), 2009. extended abstract in STOC'05.

[Wat12] Brent Waters. Functional encryption for regular languages. In *Crypto*, 2012.

8 Appendix

8.1 Additional Attack on [GVW15] using 1-keys.

We briefly describe an additional attack that holds in the [GVW15] construction, which does not apply to the [AFV11] construction. The construction of [GVW15] uses the "lazy OR" trick to handle FHE decryption. This means that instead of revealing the threshold inner product as required by FHE decryption, it reveals the exact inner product. Note that when $C(\mathbf{x}) = 0$, the exact inner product between the FHE ciphertext for $C(\mathbf{x})$ and the FHE secret key, which is revealed by the construction, is noise γ. However, since the FHE evaluation procedure is deterministic, γ is a function of the noise terms used in the original FHE encryptions of $\mathbf{x}$, learning which lead to recovery of the FHE secret key.

In more detail, consider circuits C computing linear functions. Say that the FHE ciphertext for $\mathbf{a}$ uses noise $\boldsymbol{\mu}$. Now, if we compute the function $\langle \mathbf{v}, \mathbf{a} \rangle$ homomorphically on the FHE ciphertext, the resultant ciphertext contains noise terms $\langle \mathbf{v}, \boldsymbol{\mu} \rangle$. Note that now, if $\langle \mathbf{v}, \mathbf{a} \rangle = 0$, the 1-key enables the attacker to decrypt the FHE ciphertext and learn the *noiseless* linear equation $\gamma = \langle \mathbf{v}, \boldsymbol{\mu} \rangle$, of

which $\mathbf{v}$ is known. Using several of such equations, he can recover the noise terms $\boldsymbol{\mu}$, which in turn lead to removing the noise from FHE encryptions, which lead to a recovery of the FHE secret.

We note that the authors of [GVW15] already observe that revealing the exact inner product does not allow the construction to achieve strong attribute hiding. Here we highlight that this is not a limitation of the proof technique.

8.2 Parameters for $(1, \mathrm{poly})$-PHPE

In this section, we choose the parameters so that correctness and security of the scheme are satisfied. We must satisfy the following constraints.

1. For correctness, the final magnitude of error obtained must be below $q/4$.
2. We must choose m large enough that the trapdoor generation algorithm may be invoked (Theorem 3.7).
3. We must set the parameter s used to produce the three distributions (see Section 3.4.2) large enough so that the these are statistically indistinguishable.
4. We must choose the parameter s_D used to sample the error in $\boldsymbol{\beta}_1$ large enough so that the following equation (used to argue indistinguishability of Hybrid 1 and 2) is satisfied:

$$\mathbf{e}_1 \overset{s}{\approx} \left\{ (\mathbf{K}^*)^\intercal \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix} + \mathbf{e}'' \right\} \tag{8.1}$$

   Here $\mathbf{e}_0$ is sampled from a discrete Gaussian of width s_B and $\mathbf{e}_{\mathsf{Eval}}$ is bounded by Equation 3.5.

We choose our parameters similarly to [GVW15]. The only difference between our setting and [GVW15] is that in the latter, it is required that the final error, which is dominated by $(\mathbf{K}^*)^\intercal \begin{pmatrix} \mathbf{e}_0 \\ \mathbf{e}_{\mathsf{Eval}} \end{pmatrix} \leq B_{\mathsf{Eval}}$ (say) be slightly less than the field size q, whereas, we require that this term be flooded by $\mathbf{e}''$, which is a discrete Gaussian with parameter s_D. In order to apply Lemma 2.2 to satisfy the above, we must choose s_D so that

$$\frac{B_{\mathsf{Eval}}}{s_D} = \mathrm{negl}(\kappa)$$

If we choose our parameters analogously to [GVW15], but for circuits of depth $d' = 2d$ instead of d, then, evaluating a circuit of depth d results in $B_{\mathsf{Eval}} = 2^{O(d)} = 2^{O(n^\epsilon)}$. Now, if we choose $s_D = B^2_{\mathsf{Eval}}$, then Equation 8.1 is satisfied. Since the final error is now $O(2^{O(d')})$, correctness of the parameters holds directly from [GVW15]

Summarizing, our parameters may be chosen as in [GVW15] : choose the LWE dimension $n \geq d'(\kappa)^{1/\epsilon}$ to support circuits of depth d', set $s_B = O(\sqrt{n})$, $q = 2^{O(n^\epsilon)}$ for some constant K, $m = \Theta(n \log q)$ and $s = O(\ell t n d)^{O(d)}$.

8.3 Simulator Analysis for (Q, poly) PHPE.

Analysis of the Simulator. We now argue that the simulator is correct, via a sequence of hybrids.

The Hybrids.

Hybrid 0: The real experiment.

Hybrid 1: The real game algorithm $\mathsf{QPH.Setup}$ is replaced with $\mathsf{QPH.Setup}_1^*$, which uses the knowledge of $(\mathbf{x}, \mathbf{y})$, and $\{\widehat{C_i^*} \circ \mathsf{IP}_{\gamma_i}\}_{i\in[Q]}$ to setup the public parameters and the master key.

Hybrid 2: $\mathsf{QPH.Enc}$ is replaced by $\mathsf{QPH.Enc}_1^*$, in which $\boldsymbol{\beta}_0$ is computed as in the real world, but all other ciphertext elements are derived from it. The keygen algorithm is as in the real game.

Hybrid 3: The real game $\mathsf{QPH.KeyGen}$ is replaced with $\mathsf{QPH.KeyGen}_1^*$ where instead of using the trapdoor $\mathbf{T}$ of the matrix $\mathbf{A}$, the secret keys for 0-queries are sampled using the public trapdoor $\mathbf{T_G}$ along with the trapdoor information generated using $\mathsf{QPH.Setup}_1^*$ and the secret keys for the 1-queries $\{\widehat{C_i^*} \circ \mathsf{IP}_{\gamma_i}\}_{i\in[Q]}$ are sampled using the master key.

Hybrid 4: The encryption algorithm is changed from $\mathsf{QPH.Enc}_1^*$ to $\mathsf{QPH.Enc}_2^*$. Here, the ciphertext element $\boldsymbol{\beta}_0$ is switched to random and all other ciphertext elements are derived from it.

Hybrid 5: The algorithm $\mathsf{QPH.KeyGen}_1^*$ is replaced by $\mathsf{QPH.KeyGen}_2^*$. The $\mathsf{QPH.KeyGen}_2^*$ algorithm is similar to the real world $\mathsf{QPH.KeyGen}$ algorithm, except for the keys corresponding to $\{\widehat{C_i^*} \circ \mathsf{IP}_{\gamma_i}\}_{i\in[Q]}$

Hybrid 6: The encryption algorithm is changed from $\mathsf{QPH.Enc}_2^*$ to $\mathsf{QPH.Enc}_3^*$, in which the ciphertext elements $\{\mathbf{u}_i\}$, $\{\mathbf{v}_i\}$ are changed to random. The remaining elements $\boldsymbol{\beta}_0$ and $\boldsymbol{\beta}_{1i}$ for $i \in [k]$ are as before.

Hybrid 7: The algorithm $\mathsf{QPH.Setup}_1^*$ is replaced by $\mathsf{QPH.Setup}_2^*$. The algorithm $\mathsf{PH.Setup}_2^*$ is similar to the real world $\mathsf{QPH.Setup}$ algorithm, except for how the matrices $\{\mathbf{A}_i\}_{i\in[\ell]}$ and $\{\mathbf{P}_j\}_{j\in[k]}$ is generated. This is the simulated world.

We now describe the auxiliary evaluation algorithms.

$\mathsf{QPH.Setup}_1^*\big(1^\kappa, 1^d, \mathbf{x}, \mathbf{y}, \{\widehat{C_i^*} \circ \mathsf{IP}_\gamma\}_{i\in[Q]}\big)$: Do the following:

1. Sample a matrix with associated trapdoor:

$$(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^m, 1^n, q)$$

   Let $\mathbf{G}$ be the primitive matrix with public trapdoor $\mathbf{T_G}$ that we defined in Section 3.4.1.
2. Let $\mathbf{A}_i = \mathbf{A} \cdot \mathbf{R}_i - \mathbf{y}[i] \cdot \mathbf{G} \in \mathbb{Z}_q^{n\times m}$ for $i \in [\ell]$, where $\mathbf{R}_i \leftarrow \{-1, 1\}^{m\times m}$.
3. Let $\mathbf{B}_i = \mathbf{A} \cdot \mathbf{R}'_i - \mathbf{x}[i] \cdot \mathbf{G} \in \mathbb{Z}_q^{n\times m}$ for $i \in [t]$, where $\mathbf{R}'_i \leftarrow \{-1, 1\}^{m\times m}$.
4. Generate $\mathbf{P}_1, \ldots, \mathbf{P}_k$ as follows:
   (a) For $i \in [Q]$ compute

$$\mathbf{A}_{\widehat{C_i^*}\circ\mathsf{IP}} = \mathsf{Eval}_{\mathsf{PK}}\big(\{\mathbf{A}_j\}_{j\in[\ell]}, \{\mathbf{B}_j\}_{j\in[t]}, \widehat{C_i^*} \circ \mathsf{IP}\big)$$
$$\mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}} = \mathsf{Eval}_{\mathsf{R}}\big(\{\mathbf{R}_i\},\ \{\mathbf{R}'_i\},\ \{\mathbf{B}_i\},\ \mathbf{y},\ \widehat{C_i^*} \circ \mathsf{IP}\big)$$

(b) Next, sample $\mathbf{K}_i^* \leftarrow \left(\mathcal{D}_{\mathbb{Z}^{2m},s}\right)^m$ and set:

$$\begin{aligned}\mathbf{P}_i^* &= [\mathbf{A} \mid \mathbf{A}_{\widehat{C_i^*}\circ\mathsf{IP}} + \gamma_i \cdot \mathbf{G}]\ \mathbf{K}_i^* \quad \forall i \in [Q] \\ &= [\mathbf{A} \mid \mathbf{A}\mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}}]\ \mathbf{K}_i^*\end{aligned}$$

where $\mathbf{K}_i^* = \begin{pmatrix} \mathbf{K}_{1i}^* \\ \mathbf{K}_{2i}^* \end{pmatrix}$.

(c) Choose Q random binary vectors $\mathbf{r}_1, \ldots, \mathbf{r}_Q \in \{0,1\}^k$ with hamming weight v.

(d) For $i \in [Q]$, write down the following equations in variables $\mathbf{K}_1, \ldots, \mathbf{K}_k$:

$$\mathbf{A}\ \mathbf{K}_{1i}^* + \mathbf{A}\ \mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}}\ \mathbf{K}_{2i}^* = \mathbf{A}\left(\sum_{j\in[k]} \mathbf{r}_i[j]\ \mathbf{K}_j\right) \tag{8.2}$$

Now, to satisfy the above, it suffices to choose $\mathbf{K}_1, \ldots, \mathbf{K}_k$ such that:

$$\mathbf{K}_{1i}^* + \mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}}\ \mathbf{K}_{2i}^* = \sum_{j\in[k]} \mathbf{r}_i[j]\ \mathbf{K}_j \tag{8.3}$$

(e) Note that by cover-freeness, we have that for each equation $i \in [Q]$, the RHS contains at least one matrix $\mathbf{K}_{i'}$ which does not appear in any other equation. Rearranging terms, we get for $i \in [Q]$,

$$\mathbf{K}_{i'} = \Big(\sum_{j:j\neq i'} \mathbf{r}_i[j]\ \mathbf{K}_j\Big) - \left(\mathbf{K}_{1i}^* + \mathbf{R}_{\widehat{C_i^*}\circ\mathsf{IP}}\ \mathbf{K}_{2i}^*\right) \tag{8.4}$$

(f) Sample $\mathbf{K}_j \leftarrow \mathcal{D}_{\mathbb{Z}^m,s}$ where $j \neq i'$, where $j \in [k]$, $i' \in [Q]$. Intuitvely, these are the matrices that appear in the RHS of the above equations. Set the Q values $\mathbf{K}_{i'}$ to satisfy the above equations.

(g) For $i \in [k]$, set

$$\mathbf{P}_i = \mathbf{A}\ \mathbf{K}_i \mod q$$

5. Output the master public key as $\mathsf{PH.PK} = \left(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{B}_i\}_{i\in[t]}, \mathbf{A}, \{\ \mathbf{P}_i\ \}_{i\in[k]}\right)$ and the master secret key as

$$\mathsf{PH.MSK} = \Big(\ \mathbf{T}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}'_i\}_{i\in[t]}, \{\mathbf{K}_i^*\}_{i\in[k]}\ \Big)$$

$\mathsf{PH.Enc}_1^*\big(\ \mathsf{PH.PK}, \mathsf{PH.MSK}, (\mathbf{x},\mathbf{y}), \mu, \{\ \widehat{C}_i^* \circ \mathsf{IP}_\gamma\ \}_{i\in[Q]}\ \big)$: All ciphertext elements are derived from $\boldsymbol{\beta}_0$ as follows:

1. Sample $\mathbf{s}, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^n,s_B}$ and set $\boldsymbol{\beta}_0 = \mathbf{A}^\mathsf{T}\mathbf{s} + \mathbf{e}$.
2. For $i \in [\ell]$, $j \in [t]$ compute

$$\mathbf{u}_i = \mathbf{R}_i^\mathsf{T} \cdot \boldsymbol{\beta}_0, \quad \mathbf{v}_j = (\mathbf{R}'_j)^\mathsf{T} \cdot \boldsymbol{\beta}_0$$

3. Sample $\mathbf{e}''_j \leftarrow \mathcal{D}_{\mathbb{Z}^m,s_D}$ for $j \in [k]$.
4. Set $\boldsymbol{\beta}_{1j} = (\mathbf{K}_j)^\mathsf{T}\boldsymbol{\beta}_0 + \mathbf{e}''_j + \mathbf{b}$ where $\mathbf{b} = [0\ldots,0,\lceil q/2\rceil\mu]^\mathsf{T} \in \mathbb{Z}_q^m$.
5. Output $\Big(\{\mathbf{u}_i\}, \{\mathbf{v}_i\}, \mathbf{y}, \boldsymbol{\beta}_0, \{\boldsymbol{\beta}_{1j}\}\Big)$

$\mathsf{QPH.KeyGen}_1^*(\mathsf{PH.MSK}, \widehat{C} \circ \mathsf{IP}_\rho, \{\widehat{C}_i^* \circ \mathsf{IP}_{\gamma_i}\}_{i\in[k]})$: Do the following:

1. Compute the homomorphic public key corresponding to the circuit $\widehat{C} \circ \mathsf{IP}$ as
$$\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} \leftarrow \mathsf{Eval}_{\mathsf{PK}}\big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \widehat{C} \circ \mathsf{IP}\big)$$
2. By section 3.4.3, we have that $\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} = \mathbf{A}\cdot\mathbf{R}_{\widehat{C}\circ\mathsf{IP}} - \langle \mathbf{x}, \widehat{C}(\mathbf{y})\rangle \cdot \mathbf{G}$. An admissible adversary can request:
   - A circuit $\widehat{C} \circ \mathsf{IP}_\rho$ such that $\langle \mathbf{x}, \widehat{C}(\mathbf{y})\rangle \neq \rho$, hence $\widehat{C} \circ \mathsf{IP}_\rho(\mathbf{x}, \mathbf{y}) = 0$. In this case, we have:
   $$[\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \rho \cdot \mathbf{G}] = [\mathbf{A} \cdot \mathbf{R}_{\widehat{C}\circ\mathsf{IP}} + (\rho - \langle \mathbf{x}, \widehat{C}(\mathbf{y})\rangle) \cdot \mathbf{G}]$$
   Hence, we may sample $\mathbf{K} \leftarrow \mathsf{SampleRight}\big(\mathbf{A},\ (\rho - \langle \mathbf{x}, \widehat{C}(\mathbf{y})\rangle)\cdot\mathbf{G},\ \mathbf{R}_{\widehat{C}\circ\mathsf{IP}},\ \mathbf{T}_{\mathbf{G}},\ \mathbf{P},\ s\big)$ so that:
   $$[\mathbf{A} \mid \mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \rho \cdot \mathbf{G}]\ \mathbf{K} = \mathbf{P} \pmod q$$
   Return $\mathbf{K}$.
   - The circuit $\widehat{C}_i^* \circ \mathsf{IP}_{\gamma_i}$, such that $\langle \mathbf{x}, \widehat{C}_i^*(\mathbf{y})\rangle = \gamma_i$ in which case we return $\mathbf{K}_i^*$.

$\mathsf{QPH.Enc}_2^*$: Here, the ciphertext $\boldsymbol{\beta}_0$ is switched to random. All the other ciphertext elements are derived from it as in $\mathsf{QPH.Enc}_1^*$

$\mathsf{QPH.KeyGen}_2^*$: Here, the 0 keys are chosen as in the real world, and the 1 keys are chosen as in $\mathsf{QPH.KeyGen}_1^*$.

$\mathsf{QPH.Enc}_3^*$: Here, the ciphertext components $\{\mathbf{u}_i\}$, $\{\mathbf{v}_j\}$ for $i \in [\ell]$, $j \in [t]$ are changed to random.

$\mathsf{QPH.Setup}_2^*$: Here, the public key components $\{\mathbf{B}_j\}$, $\mathbf{A}$ are chosen as in the real world, and $\mathbf{A}_i$, $\mathbf{P}_i$ are chosen as in the simulation.

We now show that consecutive hybrids are indistinguishable.

Lemma 8.1. *Hybrid 0 and Hybrid 1 are statistically indistinguishable.*

Proof. The arguments for the public parameters $\{\ \mathbf{A}_i\ \}$ and $\{\ \mathbf{B}_j\ \}$ are as in Lemma 5.2. The remaining thing to argue is the distribution of $\mathbf{P}_1, \ldots, \mathbf{P}_k$.

In Hybrid 0, $\mathbf{P}_1, \ldots, \mathbf{P}_k$ are sampled uniformly, $\mathbf{r}_1, \ldots, \mathbf{r}_Q$ are sampled randomly from $\{0,1\}^k$ such that hamming weight of each $\mathbf{r}_i$ is v, and $\mathbf{P}_1^*, \ldots, \mathbf{P}_Q^*$ are set as

$$\mathbf{P}_i^* = \sum_{j\in[k]} \mathbf{r}_i[j]\ \mathbf{P}_j$$

By the cover free property of $\mathbf{r}_1, \ldots, \mathbf{r}_Q$, it holds that each $\mathbf{r}_i$ has at least one index, say i' so that $\mathbf{r}_i[i'] = 1$ and $\mathbf{r}_j[i'] = 0$ for all $j \neq i$. Hence for $i \in [Q]$,

$$\mathbf{P}_i^* = \mathbf{P}_i' + \sum_{j\neq i'} \mathbf{r}_i[j]\ \mathbf{P}_j$$

Let I denote the set $\{i'\}_{i\in[Q]}$. Thus, in Hybrid 0, we sample $\mathbf{P}_j$ for $j \in [k] \setminus I$, sample $\mathbf{P}_j$ for $j \in [I]$ uniformly and set $\mathbf{P}_i^*$ for $i \in [Q]$ according to the above.

Now, consider Hybrid 1. Note that the equation 8.4 implies that

$$\mathbf{A} \cdot \mathbf{K}_{i'} = \mathbf{A} \cdot \left(\mathbf{K}^*_{1i} + \mathbf{R}_{\widehat{C^*_i} \circ \mathsf{IP}} \, \mathbf{K}^*_{2i} \right) - \mathbf{A} \cdot \Big(\sum_{j : j \neq i'} \mathbf{r}_i[j] \, \mathbf{K}_j \Big)$$

$$\text{Hence} \quad \mathbf{P}_{i'} = \mathbf{P}^*_i - \sum_{j : j \neq i'} \mathbf{r}_i[j] \mathbf{P}_j$$

In Hybrid 2, we sample $\mathbf{P}^*_1, \ldots, \mathbf{P}^*_Q$ as well as $\mathbf{P}_j$ for $j \in [k] \setminus I$ uniformly and compute $\mathbf{P}_j$ for $j \in [I]$. Since there is a 1-1 correspondence between $\mathbf{P}_{i'}$ and $\mathbf{P}^*_i$, it does not matter whether we pick $\mathbf{P}_{i'}$ randomly and compute $\mathbf{P}^*_i$ or pick the latter randomly and compute the former. Thus, it follows that the two distributions are statistically indistinguishable. □

Lemma 8.2. *Hybrids 1 and 2 are statistically indistinguishable.*

Proof. The difference between Hybrids 1 and 2 is in how the ciphertext elements are generated. In Hybrid 1, they are as in the real world, whereas in Hybrid 2, $\boldsymbol{\beta}_0$ is computed as in the real world, and $\{\mathbf{u}_i\}_{i \in [\ell]}$, $\{\mathbf{v}_j\}_{j \in [t]}$ and $\{\boldsymbol{\beta}_{1i}\}_{i \in [k]}$ are derived from $\boldsymbol{\beta}_0$. The analysis of the elements $\{\mathbf{u}_i\}_{i \in [\ell]}$, $\{\mathbf{v}_j\}_{j \in [t]}$ is exactly as in Lemma 5.3. It remains to argue about the distribution of $\{\boldsymbol{\beta}_{1i}\}_{i \in [k]}$, which we do here.

Now, in Hybrid 1, we have:

$$\boldsymbol{\beta}_{1i} = \mathbf{P}_i^\mathsf{T} \mathbf{s} + \mathbf{e}'_i + \mathbf{b}$$

In Hybrid 1, we have

$$\boldsymbol{\beta}_{1i} = (\mathbf{K}_i)^\mathsf{T} \boldsymbol{\beta}_0 + \mathbf{e}''_i + \mathbf{b}$$

Note that $\mathbf{A} \, \mathbf{K}_i = \mathbf{P}_i \mod q$ and $\boldsymbol{\beta}_0 = \mathbf{A}^\mathsf{T} \mathbf{s} + \mathbf{e}$, hence

$$(\mathbf{K}_i)^\mathsf{T} \boldsymbol{\beta}_0 + \mathbf{e}''_i + \mathbf{b} = \mathbf{P}_i^\mathsf{T} \mathbf{s} + \mathbf{b} + \left(\mathbf{K}^\mathsf{T} \mathbf{e} + \mathbf{e}''_i \right)$$

By choosing

$$\mathbf{e}'_i \overset{s}{\approx} \left(\mathbf{K}^\mathsf{T} \mathbf{e} + \mathbf{e}''_i \right)$$

we have that the two worlds are statistically indistinguishable. □

Lemma 8.3. *Hybrids 2 and 3 are statistically indistinguishable.*

Proof. For the case of 0-queries, the argument is the same as in Lemma 5.3. For the case of 1-queries, note that in Hybrid 1, $\mathbf{P}^*_i$ are sampled uniformly, and $\mathbf{K}^*_i$ are sampled using $\mathsf{SampleLeft}$ to satisfy:

$$[\mathbf{A} \mid \mathbf{A}_{\widehat{C^*_i} \circ \mathsf{IP}} + \gamma_i \cdot \mathbf{G}] \, \mathbf{K}^*_i = \mathbf{P}^*_i \pmod q$$

whereas in Hybrid 2, $\mathbf{K}^*_i$ are sampled as $(\mathcal{D}_{\mathbb{Z}^{2m}, s})^m$ and $\mathbf{P}^*$ are set as

$$\mathbf{P}^*_i = [\mathbf{A} \mid \mathbf{A}_{\widehat{C^*_i} \circ \mathsf{IP}} + \gamma_i \cdot \mathbf{G}] \, \mathbf{K}^*_i \pmod q$$

By Section 3.4.2, these two distributions are statistically indistinguishable. □

Indistinguishability between Hybrids 3 and 4 follows from LWE exactly as in Lemma 5.5, between Hybrids 4 and 5 is analogous to the argument in Lemma 8.3, between Hybrids 5 and 6 is as in Lemma 5.7 and between Hybrids 6 and 7 is as in Lemma 8.1.

8.4 Full Security for Single Key Linear FE

Definition 8.4 (FULL-SIM security)**.** Let FE be a single key functional encryption scheme for a circuit family $\mathcal{C}$. For every p.p.t. adversary Adv and a stateful p.p.t. simulator Sim, consider the following two experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},A}(1^\kappa)\textbf{:}}$

1: $(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\kappa)$
2: $(\mathbf{a}, \mathsf{st}) \leftarrow A_1^{\mathsf{FE.Keygen}(\mathsf{MSK},\cdot)}(\mathsf{PK})$
3: $\mathsf{CT} \leftarrow \mathsf{FE.Enc}(\mathsf{PK}, \mathbf{a})$
4: $\alpha \leftarrow A_2(\mathsf{CT}, \mathsf{st})$
5: Output $(\mathbf{a}, \alpha)$

$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}(1^\kappa)\textbf{:}}$

1: $\mathsf{PK} \leftarrow \mathsf{Sim}(1^\kappa)$
2: $(\mathbf{a}, \mathsf{st}) \leftarrow A_1^{\mathsf{Sim}(\cdot)}(\mathsf{PK})$
3: $\mathsf{CT} \leftarrow \mathsf{Sim}\big(\, C, C(\mathbf{a}) \,\big)$
4: $\alpha \leftarrow A_2(\mathsf{CT}, \mathsf{st})$
5: Output $(\mathbf{a}, \mu, \alpha)$

The functional encryption scheme FE is then said to be FULL-SIM-secure if there is a stateful p.p.t. simulator Sim such that for every p.p.t. adversary $A = (A_1, A_2)$, the following two distributions are computationally indistinguishable.

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},A}(1^\kappa)\right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}(1^\kappa)\right\}_{\kappa\in\mathbb{N}}$$