Optimal Bounded-Collusion Secure Functional Encryption

Prabhanjan Ananth[*]
CSAIL, MIT

Vinod Vaikuntanathan[†]
CSAIL, MIT

Abstract

We construct private-key and public-key functional encryption schemes secure against adversaries that corrupt an a-priori bounded number of users and obtain their functional keys, from minimal assumptions.

For a collusion bound of $Q = Q(\lambda)$ (where λ is the security parameter), our public-key (resp. private-key) functional encryption scheme (a) supports the class of all polynomial-size circuits; (b) can be built solely from a vanilla public-key (resp. private-key) encryption scheme; and (c) has ciphertexts that grow linearly with the collusion bound Q. Previous constructions were sub-optimal with respect to one or more of the above properties. The first two of these properties are the best possible and any improvement in the third property, namely the ciphertext size dependence on the collusion bound Q, can be used to realize an indistinguishability obfuscation scheme.

In addition, our schemes are adaptively secure and make black-box use of the underlying cryptographic primitives.

1 Introduction

Functional Encryption [29, 12] (FE) is a powerful type of encryption where the owner of a secret key sk can generate special-purpose functional secret keys sk_F which allow anyone to compute $F(x)$ given an encryption of x. The standard and demanding security notion for functional encryption is collusion-resistance which, informally stated, requires that an adversary who holds functional secret keys for an *arbitrary polynomial* number of functions $F_1, F_2, \ldots, F_m$ of her choice should learn no more than $F_1(x), F_2(x), \ldots, F_m(x)$ given an encryption of x. Collusion-resistant functional encryption schemes are extremely powerful: [4, 11, 5] show that such FE schemes can be used to construct indistinguishability obfuscators and therefore, can be used to instantiate a vast majority of cryptographic primitives (see [30] and a large number of followup works.) It is no surprise then that collusion-resistant FE schemes are very hard to construct and indeed, to this date, we do not know constructions from standard cryptographic assumptions.

In practical uses of functional encryption, however, the weaker notion of *bounded collusion-resistance* might suffice. Bounded collusion-resistance permits the secret-key owner to release an unbounded number of functional keys as before, but ensures security only against adversaries that

[*]`prabhanjan@csail.mit.edu`
[†]`vinodv@mit.edu`

corrupt an a-priori bounded (polynomial) number $Q = Q(\lambda)$ of functional keys. (Here and henceforth, λ denotes the security parameter.) As pointed out in [24, 2], bounded collusion-resistance is analogous to proving security under the assumption that not too many users are corrupted, which is widely accepted as reasonable in protocol design. Bounded collusion-resistance is appropriate in scenarios where functional keys are tied to users, and a large colluding set of users is hard to form. Technically, bounded-collusion resistance has been well-studied with the goals of improving efficiency, reducing computational assumptions, and supporting a larger class of functions; see [16, 25, 15, 23, 28, 24, 26, 2, 1] and the references therein.

1.1 Prior Work on Bounded-Collusion FE

Early work by Dodis, Katz, Xu and Yung [16] showed how to construct a Q-bounded identity-based encryption (IBE) scheme, a particularly simple form of FE, where the public parameters had size $O(Q^2\lambda)$, and the ciphertexts and secret keys had size $O(Q\lambda)$, starting from any public-key encryption scheme. Goldwasser, Lewko and Wilson [23] later showed a construction with public parameters of size $O(Q\lambda)$, and ciphertexts and secret keys of size $O(\lambda)$, albeit under more structured algebraic assumptions. More recently, Döttling and Garg [18] and followup works [13, 17] showed how to bootstrap any bounded collusion IBE with public parameters of size $Q^{1-\epsilon} \cdot \mathsf{poly}(\lambda)$, irrespective of ciphertext and secret-key length, into a full-fledged (i.e., fully collusion-resistant) IBE scheme. This gives us a dichotomy for IBE: Q-bounded IBE with public parameters of size $\Omega(Q)$ exists under the minimal assumption of public-key encryption; and doing any better in terms of the size of public parameters is as hard as achieving unbounded-collusion IBE.

In the other extreme, the situation with general functional encryption (FE) is less clear-cut. We do have bootstrapping theorems that come from the works of Ananth and Jain [4, 5], and that of Bitansky and Vaikuntanathan [11], which tell us that FE with *ciphertexts* of size $Q^{1-\epsilon} \cdot \mathsf{poly}(\lambda)$ (for powerful enough function classes, roughly speaking, ones that can compute weak pseudorandom functions) can be transformed into an indistinguishability obfuscator which in turn gives us a fully collusion-resistant FE scheme [30].

It is the other side of the bargain that is less clear-cut. Zooming in on the dependence of the ciphertext size on the collusion bound, we have the result of Gorbunov, Vaikuntanathan and Wee [24], who built on the work of Sahai and Seyalioglu [28] and showed a Q-bounded FE scheme for *Boolean functions*[1] computable in NC^1 from any public-key encryption scheme; the ciphertexts in the scheme had size $Q^4 \cdot \mathsf{poly}(\lambda)$. They also showed how to extend this to support all poly-time computable functions, but at the expense of an additional assumption, namely pseudorandom functions that can be computed in NC^1, an object that can be constructed under algebraic assumptions such as factoring, DDH and LWE. (See Figure 1 for a detailed comparison.) Agrawal and Rosen [2] showed how to reduce the ciphertext size to $Q^2 \cdot \mathsf{poly}(\lambda)$ under the LWE assumption. Chen, Vaikuntanathan, Waters, Wee and Wichs [14] very recently showed how to reduce the dependence even further to $Q \cdot \mathsf{poly}(\lambda)$ under the LWE assumption, except they could only achieve private-key FE. The ciphertext size dependence on Q in this last result is the best possible (without constructing IO) except that (a) they rely on LWE; and (b) they only achieve private-key FE. Even in the much weaker setting of public-key *attribute-based* encryption (ABE), the best known ciphertext size is $Q^2 \cdot \mathsf{poly}(\lambda)$ in constructions that rely only on public-key encryption [26].

[1]Handling functions with output size ℓ is morally the same as increasing the collusion bound and handling ℓ functions with Boolean output. Indeed, this is made precise in the results of [11, 5, 20, 27].

1.2 Our Results

In short, the situation with constructions of bounded-collusion FE is complex and in particular, the question of achieving "optimal" FE remains open. That is:

> *Can we construct Q-bounded FE for all functions in* $\mathsf{P/poly}$ *with ciphertexts of size* $Q \cdot \mathsf{poly}(\lambda)$ *starting from any public-key encryption scheme?*

This is "optimal" because (a) in terms of functionality, $\mathsf{P/poly}$ is the best one could do; (b) in terms of assumptions, public-key encryption is minimal; and (c) in terms of efficiency or ciphertext size, doing any better than $O(Q)$ would take us all the way, giving us full-fledged collusion-resistant FE (and even IO).

Our main result in this paper is precisely such an "optimal" FE scheme. The scheme has several additional features. First, it is adaptively secure, namely, it will allow the adversary to make the secret-key queries and encryption challenge fully adaptively. Second, it is black-box, namely, the FE construction will make black-box use of the underlying cryptographic primitives. Third, we show a version of our construction in the private-key FE setting where our only assumption will be the existence of private-key encryption schemes (which is equivalent to assuming that one-way functions exist.) Fourth and finally, our techniques are elementary, as we will describe in Section 1.3.

Main Theorem. Assuming the existence of public-key encryption schemes, for any $Q = Q(\lambda)$, there is a Q-bounded FE scheme with ciphertexts of size $Q \cdot \mathsf{poly}(s, \lambda)$ for all functions in $\mathsf{P/poly}$ with circuit size s.

	Ciphertext Size	Circuit Class	Assumptions	Remarks
[24]	$Q^4\text{poly}(\lambda, s)$	NC^1	PKE	Public-Key, Adaptive
	$Q^4\text{poly}(\lambda, s)$	NC^1	OWFs	Private-Key, Adaptive
	$Q^4\text{poly}(\lambda, s)$	$\mathsf{P/Poly}$	DDH/LWE	Public-Key, Adaptive
[2]	$Q^2 + \text{poly}(\lambda, s)$	$\mathsf{P/Poly}$	LWE/Ring-LWE	Public-Key, Selective
[14]	$Q \cdot \text{poly}(\lambda, s)$	$\mathsf{P/Poly}$	LWE	Private-Key, Selective
Our Work	$Q \cdot \text{poly}(\lambda, s)$	$\mathsf{P/Poly}$	PKE	Public-Key, Adaptive
	$Q \cdot \text{poly}(\lambda, s)$	$\mathsf{P/Poly}$	OWFs	Private-Key, Adaptive

Figure 1: State of the art for bounded key functional encryption schemes in terms of query dependence. Q denotes the number of circuit queries allowed in the security experiment and s denotes the size of the circuits for which functional keys are issued.

We do caution the reader that our focus will be on the dependence of the ciphertext size on the collusion-bound Q. Ciphertexts in our scheme grow with the circuit-size of the functions that the scheme supports (denoted s in Figure 1). On the one hand, for constructions that rely only on the minimal assumption of public-key encryption, this dependence seems hard to remove; indeed, even the best 1-bounded FE with ciphertext size sublinear in the circuit-size of the (Boolean) functions assumes (subexponential) LWE [22]. On the other hand, we show how to translate any improvement in this state of affairs for 1-bounded FE into a corresponding improvement in Q-bounded FE with ciphertexts that grow *linearly* in the collusion bound Q. Concretely, applying our techniques to the 1-bounded FE of [22] gives us a Q-bounded FE from subexponential LWE where ciphertexts grow as $Q \cdot \mathsf{poly}(\lambda, d)$ where d is the circuit-depth, improving on [2] (who achieve a quadratic dependence in Q) and on [14] (who construct a private-key FE scheme with a linear dependence in Q).

1.3 Technical Overview

We give an overview of the techniques used in proving our result. For the current discussion, our focus will be on the public-key setting; the techniques carry over *mutatis mutandis* to the private-key setting as well. We show our result in two steps. In the first step, we construct a public-key bounded-key FE for $\mathsf{P/Poly}$ starting from any public-key encryption scheme. We will not worry about optimizing the ciphertext size; indeed, it will be a large polynomial in the collusion bound Q. In the second step, we show a general way to reduce the ciphertext size: we show how to transform an FE scheme, where the ciphertext complexity grows polynomial in the collusion bound, into a FE scheme with *linear complexity*.

We now describe an overview of the techniques involved in the two steps, in order. In the technical sections, we invert the order of presentation since the second step (see section 4) is simpler than the first (see section 5).

1.3.1 First Step: Bounded-Key FE for $\mathsf{P/Poly}$.

Our starting point is the observation from [28, 24] that secure multiparty computation protocols with certain properties can be used to construct FE schemes; for [28], it was Yao's two-party computation protocol [31] and for [24], it was a non-interactive version of the BGW multi-party protocol [9]. Broadly speaking, our goal in this paper is to identify *the right* notion of MPC that can be turned into *optimal* bounded-collusion FE.

Towards this end, we define secure multiparty computation protocols in a client-server framework where there is a single client who wishes to delegate an a-priori bounded number Q of computations to N servers. We first describe the syntax of such protocols and then the security we require of them. A protocol in the client-server framework proceeds in two phases:

- An *offline phase* where the client encodes a private input x into N encodings, and the i^{th} server gets the i^{th} encoding.
- An *online phase* which is executed Q times, once for every function that the client wishes to delegate. In the j^{th} session, the client encodes a circuit C_j into N encodings, and sends each server an encoding. At this stage, only n of the N servers come online, perform some local computation on their encodings, and output a single message each. (We call the local computation function Local.) A public decoding algorithm can then reconstruct the value $C_j(x)$ from these server messages. (We call the reconstruction function Decode.)

  Crucially, we require that the client does not keep any shared state between the online and offline phases.

As for security, we consider an adversary that corrupts an arbitrary size-t subset of the servers (for some pre-determined t) and learns (a) the offline phase messages received by these t servers and (b) *all* the messages in the online phase. We require that such an adversary does not learn anything more about the client input x other than $\{C_j(x)\}_{j\in[Q]}$. This requirement is captured through a simulation-based definition. Two aspects make it challenging to construct such protocols:

- *Reusability*: the input encodings generated by the client should be reusable across different computations; and

- *Dynamic Recovery*: the ability for only a subset of servers to come together in the online phase to recover the output.

For the current discussion, we call secure protocols that satisfy both the above properties as *reusable dynamic MPC* protocols. In the technical sections, we will not explicitly use the terminology of reusable dynamic MPC protocols and just refer to them as client-server protocols.

Implicit in [24] is a construction of a reusable dynamic MPC protocol, where the circuits delegated by the client are in NC^1. There is a fundamental barrier in extending their approach to handle circuits in $\mathsf{P/Poly}$ as they crucially use a two-round MPC protocol (derived from BGW) that securely computes polynomials. Circuits in $\mathsf{P/Poly}$ are believed to not have efficient polynomial representations. While several recent works [10, 21, 3, 19, 6] demonstrate two-round MPC protocols that securely compute $\mathsf{P/Poly}$, they fail to simultaneously satisfy reusability *and* dynamic recovery. Nonetheless, we will crucially use the construction of reusable dynamic MPC protocol for NC^1 [24], denoted by Π_{NC^1}, to build a protocol for $\mathsf{P/Poly}$.

From Client-Server Protocol for $\mathsf{P/Poly}$ to Bounded-Key FE for $\mathsf{P/Poly}$. Before we construct reusable dynamic MPC protocols for $\mathsf{P/Poly}$, we first show how such protocols are useful in obtaining bounded-collusion FE for $\mathsf{P/Poly}$. As an intermediate tool, we use a single-key FE scheme for $\mathsf{P/Poly}$. We call such a scheme $\mathsf{1fe}$ and we denote the bounded-collusion FE scheme that we wish to construct to be BFE. The construction of BFE, which follows along the lines of [24], proceeds as follows:

- The setup of BFE invokes $N = \mathrm{poly}(Q)$ instantiations of $\mathsf{1fe}$. The N public keys of $\mathsf{1fe}$ form the master public key of BFE and similarly the N secret keys of $\mathsf{1fe}$ form the master secret key of BFE.

- To encrypt an input x in BFE, run the offline phase of the client-server framework. Denote the output to be $(\widehat{x}^1, \ldots, \widehat{x}^N)$. Encrypt $\widehat{x}^{\boldsymbol{u}}$ under the $\boldsymbol{u}^{th}$ instantiation of $\mathsf{1fe}$. Output all the N ciphertexts of $\mathsf{1fe}$.

- The key generation for a circuit C in BFE is done as follows: run the client delegation procedure CktEnc on C to obtain $(\widehat{C}^1, \ldots, \widehat{C}^N)$. Pick a random n-sized subset $\mathbf{S} \subseteq [N]$ and generate $\mathsf{1fe}$ functional keys for $\mathsf{Local}(\widehat{C}^{\boldsymbol{u}}, \cdot)$ (recall that Local is part of the online phase in client-server framework) for every $\boldsymbol{u}$ in the set $\mathbf{S}$. Output all the n functional keys of $\mathsf{1fe}$.

  Note that here we crucially use the fact that the client does not share state between the offline and online phases.

- The decryption proceeds by first decrypting the $\boldsymbol{u}^{th}$ ciphertext of $\mathsf{1fe}$ using the $\boldsymbol{u}^{th}$ functional key to obtain the encoding $\widehat{y}^{\boldsymbol{u}}$. Then run Decode to recover the answer.

The correctness of BFE follows from the correctness guarantees of $\mathsf{1fe}$ and the reusable dynamic MPC framework. To argue security, as in [24], a simple combinatorial argument is first invoked to prove that the size of pairwise intersections of the sets chosen during the key-generation procedures of all the Q functional keys is at most t. For this argument to work, we need to set N to be a sufficiently large polynomial in Q. Using this observation, we can deduce that at most t

instantiations of $\mathsf{1fe}$ can be rendered insecure. An $\mathsf{1fe}$ instantiation being rendered insecure means that the corresponding server is corrupted in the client-server framework; note that there is a one-to-one correspondence between the number of instantiations of $\mathsf{1fe}$ and the number of servers in the client-server framework. We can then use the property that the client-server protocol is secure even if at most t servers are corrupted, to argue that the scheme BFE is secure.

Moreover, since $\mathsf{1fe}$ can be based on public-key encryption (resp., one-way functions), we obtain a public-key (resp., secret-key) BFE for $\mathsf{P/Poly}$ from reusable dynamic MPC for $\mathsf{P/Poly}$ assuming only public-key encryption (resp., one-way functions).

Reusable Dynamic MPC Protocol for $\mathsf{P/Poly}$. Now that we have shown that reusable dynamic MPC is useful for constructing bounded-key FE, we shift our focus to building this object.

Towards this, we first define the abstraction of *correlated garbling*. This abstraction allows for generating multiple garbled circuits from a shared random string. More specifically, it comprises of two algorithms: $\mathsf{CorrGarb}$ and $\mathsf{CorrEval}$. The correlated garbling algorithm $\mathsf{CorrGarb}$ takes as input circuit C, input x, a random string R (not necessarily uniformly generated) and outputs a garbled circuit GC and appropriate wire labels $\mathbf{K}_x$. The evaluation algorithm $\mathsf{CorrEval}$ takes as input $(\mathsf{GC}, \mathbf{K}_x)$ and outputs $C(x)$. We require that all the different correlated garbled circuits $\{\mathsf{GC}_i \leftarrow \mathsf{CorrGarb}(C_i, x, R)$ produced using the *same string* R do not reveal any information about x beyond $\{(C_i, C_i(x))\}$.

We use this abstraction to transform Π_{NC^1} (recall, Π_{NC^1} is a reusable dynamic protocol for NC^1) into a reusable dynamic for $\mathsf{P/Poly}$ as follows:

- Offline Phase: to encode an input x, generate a random string R (as dictated by correlated garbling) and then encode (x, R) using the offline phase of Π_{NC^1} to obtain N input encodings.
- Online Phase: in the i^{th} session, let C_i be the circuit delegated by the client. The client generates the online phase of Π_{NC^1} on the circuit $\mathsf{CorrGarb}(C_i, \cdot, \cdot)$ to obtain N circuit encodings and sends one encoding to each of the servers. A subset of the servers perform local computation of Π_{NC^1} and each of them output a single message. The value $C_i(x)$ can be recovered from the outputs of the servers in two steps: (i) run the decoding procedure of Π_{NC^1} to obtain the correlated garbled circuit-wire keys pair $(\mathsf{GC}_i, \mathbf{K}_x^i)$ of (C_i, x) and then, (ii) run $\mathsf{CorrEval}$ on the correlated garbled circuit to recover the answer.

In order to implement the above construction, it is required that $\mathsf{CorrGarb}$ is representable by an NC^1 circuit: this is because Π_{NC^1} only allows for delegating computations in NC^1. The security of the above construction follows from the fact that the different correlated garbled circuits along with wire keys $\{(\mathsf{GC}_i, \mathbf{K}_x^i)\}$ can be simulated using $\{(C_i, C_i(x))\}$: note that all the correlated garbled circuits are computed as a function of the same random string R. In Section 5, we give a direct construction of client-server protocol from correlated garbling; in particular we do not assume that a client-server protocol for NC^1 as implicitly proposed in [24].

All that remains is to construct a correlated garbling scheme with the garbling function in NC^1. We introduce novel techniques in this construction and this is the main technical contribution of the paper.

Construction of Correlated Garbling. The main hurdle in constructing a correlated garbling scheme is to ensure the security of different correlated garbled circuits computed using the same randomness. A first attempt to constructing correlated garbling is the following:

- Let s be the number of wires in the circuit to be garbled. For every wire w in the circuit, generate poly(λ, Q) number of uniformly random keys, denoted by the vector $\overrightarrow{\mathbf{K}^0_w}$, associated with bit 0 and λ number of keys $\overrightarrow{\mathbf{K}^1_w}$ for bit 1. Similarly, for every gate G in the circuit, generate poly(λ, Q) number of random strings, denoted by $\overrightarrow{\mathbf{R}_G}$. The collection of all the strings form the random string R that will be input to $\mathsf{CorrGarb}$.

- To garble a circuit C, $\mathsf{CorrGarb}$ chooses a *random* λ-sized subset S; for every wire w, it generates the wire key K^0_w (resp., K^1_w) for w by XOR-ing the subset S of keys in $\overrightarrow{\mathbf{K}^0_w}$ (resp., $\overrightarrow{\mathbf{K}^1_w}$). Similarly, generate R_G by XOR-ing the subset S of random strings in $\overrightarrow{\mathbf{R}_G}$. R_G will be used as randomness for encryption and to randomly permute the ciphertexts for the garbled table associated with G. Using the wire keys $\{K^0_w, K^1_w\}$ and the randomness $\{R_G\}$, generate a garbling of C using the garbling scheme of [31]. The output of $\mathsf{CorrGarb}$ is the garbling of C along with the wire keys associated with the input x.

- $\mathsf{CorrEval}$ is the same as the evaluation algorithm of the garbling scheme by [31].

Note that the distribution of wire keys $\{K^0_w, K^1_w\}$ as well as the random strings $\{R_G\}$ as described above is identical to uniform distribution. The string R (input to $\mathsf{CorrGarb}$) can be reused Q times to generate Q different collections of wire keys and random strings; each such collection can be generated using a different random set S. A combinatorial argument can be used to argue that the joint distribution of the Q collections of wire keys and the random strings generated using R, is identical to the product uniform distribution. This observation is crucial in proving that the above scheme is a secure candidate of correlated garbling. However, to implement $\mathsf{CorrGarb}$ algorithm in NC^1, we require algebraic assumptions; the garbling procedure of [31] is in NC^1 if the underlying encryption procedure is in NC^1 which in turn assumes PRGs in NC^1 (we need the output of the PRG to be twice as long as the length of the seed). Recall that in the construction of client-server protocol for $\mathsf{P/Poly}$, we needed a correlated garbling scheme with $\mathsf{CorrGarb}$ in NC^1.

To overcome the above barrier, we propose an alternate garbling procedure.

- Instead of invoking the PRG during the execution of $\mathsf{CorrGarb}$, we instead invoke this during the generation of R. While doing so, we observe that it is no longer necessary that PRG needs to be computable in NC^1, since there is no such restriction when generating R. As a result, we will end up generating all the keys in $\{\overrightarrow{\mathbf{K}^0_w}, \overrightarrow{\mathbf{K}^1_w}\}$ using *any* pseudorandom generator.

- To maintain correctness, we need to encrypt a subset of the seeds of the PRG as part of the garbled table. Arguing security is more challenging now. We need to argue that the joint distribution of the Q collections of the wire keys and the random strings computed using R, is identical to the product uniform distribution, *even if some of the* PRG *seeds generating the wire keys are leaked* and this step is crucial to the proof of the correlated garbling lemma (Lemma 2).

The above template is an over-simplified presentation of correlated garbling and we refer the reader to the technical sections for a precise description.

Summarizing the first step. We summarize the steps to construct a bounded-key functional encryption for $\mathsf{P/Poly}$.

1. We construct correlated garbling for $\mathsf{P/Poly}$ from one-way functions.
2. Combining correlated garbling with techniques from [24], we construct a protocol in the client-server framework (satisfying both reusability and dynamic recovery) that handles $\mathsf{P/Poly}$ computations.
3. Finally, we construct bounded-key FE for $\mathsf{P/Poly}$ from a client-server protocol and single-key functional encryption for $\mathsf{P/Poly}$ [28, 24].

The ciphertext complexity in the resulting FE scheme, however, grows polynomially in Q.

1.3.2 Second Step: Linear Dependence in Query Complexity.

In the second step, we give a generic transformation to turn the FE scheme resulting from the first step into one that satisfies linear complexity property. This transformation is remarkably simple and draws connections to the classical load balancing problem. Recall in the load balancing problem, there are Q reviewers and there are Q papers to review, with each reviewer having bandwidth to review at most q papers. Assigning papers at random to the reviewers ensures that each reviewer has to review one paper on average. By a simple Chernoff argument coupled with union bound argument, it follows that, as long as q is large enough, the probability that any reviewer has to review more than q papers is small. We propose our transformation along these lines: let bfe be the FE scheme obtained from the first step and let BFE be the FE scheme with linear complexity that we wish to construct. To tolerate a query bound Q, we consider Q instantiations of bfe in parallel, where the collusion bound (read as "load") in bfe is set to be q.

- To encrypt a message x in BFE, encrypt x in all the instantiations of bfe.
- To generate a functional key for a circuit C, pick an index i in $[Q]$ at random and generate a bfe functional key corresponding to the i^{th} instantiation. This is akin to assigning a paper to a reviewer at random.

If we set q to be security parameter, we can prove (using Chernoff and union bounds) that it is highly unlikely that the number of bfe functional keys issued for any given index is greater than q. This allows us to invoke the security of bfe scheme to prove the security of BFE. Moreover, the ciphertext complexity of BFE is linear in Q, as desired! (each bfe ciphertext is of size fixed polynomial in the security parameter and in particular, independent of Q).

2 Preliminaries

We denote the security parameter by λ. Suppose x and y be two strings. Then, we denote $x \circ y$ to be the concatenation of x and y.

Let D be a distribution with an efficient sampler. We denote the process of sampling v from D to be $v \xleftarrow{\$} D$. The statistical distance between two distributions D_0 and D_1 is ε if $\sum_{v \in V} |\mathsf{Pr}[v \xleftarrow{\$} D_0] - \mathsf{Pr}[v \xleftarrow{\$} D_1]| \leq 2\varepsilon$, where V is the support of both D_0 and D_1. Two distributions D_0 and D_1 are computationally indistinguishable if for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$, the following holds: $\left| \underset{v \xleftarrow{\$} D_0}{\mathsf{Pr}}[0 \leftarrow \mathcal{A}(v)] - \underset{v \xleftarrow{\$} D_1}{\mathsf{Pr}}[0 \leftarrow \mathcal{A}(v)] \right| \leq \mathsf{negl}(\lambda)$, for some negligible function negl.

We assume that without loss of generality, every polynomial-sized circuit considered in this work contain only boolean gates (over any universal basis) with at most two output wires. Note that if every gate in a polynomial-sized circuit has at most one output wire then this circuit is representable as a polynomial-sized formula and thus, is in NC^1. The class of all polynomial-sized circuits is denoted by $\mathsf{P/Poly}$.

2.1 Bounded-Key Functional Encryption

A public-key functional encryption scheme bfe associated with a class of boolean circuits $\mathcal{C}$ is defined by the following algorithms.

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^Q, 1^s)$: On input security parameter λ, query bound Q, maximum size of the circuits s for which functional keys are issued, output the master secret key msk and the master public key mpk.
- **Key Generation,** $\mathsf{KeyGen}(\mathsf{msk}, C)$: On input master secret key msk and a circuit $C \in \mathcal{C}$, output the functional key sk_C.
- **Encryption,** $\mathsf{Enc}(\mathsf{mpk}, x)$: On input master public key mpk, input x, output the ciphertext ct.
- **Decryption,** $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct})$: On input functional key sk_C, ciphertext ct, output the value y.

Remark 1. *A private-key functional encryption scheme is defined similarly, except that* $\mathsf{Setup}(1^\lambda, 1^Q, 1^s)$ *outputs only the master secret key* msk *and the encryption algorithm* Enc *takes as input the master secret key* msk *and the message* x.

Remark 2. *Henceforth,* Setup *will only take as input* $(1^\lambda, 1^s)$ *in the case when* $Q = 1$.

A functional encryption scheme satisfies the following properties.

Correctness. Consider an input x and a circuit $C \in \mathcal{C}$ of size s. We require the following to hold for every $Q \geq 1$:

$$\mathsf{Pr}\left[C(x) \leftarrow \mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) \ : \ \substack{(\mathsf{mpk},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^Q, 1^s);\\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C);\\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x)} \right] \geq 1 - \mathsf{negl}(\lambda),$$

for some negligible function negl.

Efficiency. $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}$ and Dec run in time polynomial in their respective inputs.

We define a measure of efficiency that captures the dependance of the ciphertext complexity on the query bound. We define this formally below.

Definition 1 (Linear Complexity)**.** *A functional encryption scheme* $\mathsf{bfe} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to have linear complexity if the following holds:*

- *The time to compute* $\mathsf{Enc}(\mathsf{mpk}, x)$ *is* $Q \cdot \mathrm{poly}(\lambda, s)$.
- *The time to compute* $\mathsf{KeyGen}(\mathsf{msk}, C)$ *for a circuit of size* s *is* $Q \cdot \mathrm{poly}(\lambda, s)$.

where $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^Q, 1^s)$.

Security. To define the security of a bounded-key functional encryption scheme bfe, we define two experiments Expt_0 and Expt_1. Experiment Expt_0, also referred to as *real* experiment, is parameterized by PPT stateful adversary $\mathcal{A}$ and challenger Ch. Experiment Expt_1, also referred to as *simulated* experiment, is parameterized by PPT adversary $\mathcal{A}$ and PPT stateful simulator Sim.

$\underline{\mathsf{Expt}_0^{\mathsf{bfe},\mathcal{A},\mathsf{Ch}}(1^\lambda)}$:

- $\mathcal{A}$ outputs the query bound Q and the maximum circuit size s.
- Ch executes $\mathsf{bfe.Setup}(1^\lambda, 1^Q, 1^s)$ to obtain the master public key-master secret key pair $(\mathsf{mpk}, \mathsf{msk})$.
- **Circuit Queries**: $\mathcal{A}$, with oracle access to $\mathsf{bfe.KeyGen}(\mathsf{msk}, \cdot)$, outputs the challenge message x.
- **Challenge Message Query**: Ch outputs the challenge ciphertext ct.
- **Circuit Queries**: $\mathcal{A}$, with oracle access to $\mathsf{bfe.KeyGen}(\mathsf{msk}, \cdot)$, outputs the bit b.
- If the total number of oracle calls made by $\mathcal{A}$ is greater than Q, output $\perp$. Otherwise, output b.

$\underline{\mathsf{Expt}_1^{\mathsf{bfe},\mathcal{A},\mathsf{Sim}}(1^\lambda)}$:

- $\mathcal{A}$ outputs the query bound Q and the maximum circuit size s.
- Sim, on input $(1^\lambda, 1^Q, 1^s)$, outputs the master public key mpk.
- **Circuit Queries**: $\mathcal{A}$, with oracle access to Sim (generating simulated functional keys), outputs the challenge message x.
  - Let QSet be the set of circuit queries made by $\mathcal{A}$ to Sim.
  - Construct the set $\mathcal{V}$ as follows: for every $C \in \mathsf{QSet}$, include $(C, C(x))$ in $\mathcal{V}$.
- **Challenge Message Query**: $\mathsf{Sim}(1^{|x|}, \mathcal{V})$ outputs the challenge ciphertext ct.
- **Circuit Queries**: $\mathcal{A}$, with oracle access to Sim (generating simulated functional keys), outputs bit b.
- If the total number of circuit queries made by $\mathcal{A}$ is greater than Q, output $\perp$. Otherwise, output b.

A public-key functional encryption scheme is adaptively secure if the output distributions of the above two experiments are computationally indistinguishable. More formally,

Definition 2 (Adaptive Security). *A public-key functional encryption scheme* bfe *is* **adaptively secure** *if for every large enough security parameter* $\lambda \in \mathbb{N}$*, every PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* Sim *such that the following holds:*

$$\left| \mathsf{Pr}\left[0 \leftarrow \mathsf{Expt}_0^{\mathsf{bfe},\mathcal{A},\mathsf{Ch}}(1^\lambda)\right] - \mathsf{Pr}\left[0 \leftarrow \mathsf{Expt}_1^{\mathsf{bfe},\mathcal{A},\mathsf{Sim}}(1^\lambda)\right] \right| \leq \mathsf{negl}(\lambda),$$

for some negligible function negl.

Remark 3. *The selective security notion can be defined by similarly formulating the real and the simulated experiments. The only difference between selective security and adaptive security notions is that in the selective security notion, the adversary is supposed to output the challenge message even before it receives the master public key or makes any circuit query.*

In the private-key setting, selective and adaptive security notions can be defined similarly.

3 Result Statements

We prove our result in two steps. In the first step, we present a transformation that converts a bounded-key functional encryption scheme, that doesn't have linear complexity property, into one that satisfies linear complexity.

Generic transformation to achieve linear complexity. We prove the following theorem in Section 4.

Theorem 1. *Consider a class $\mathcal{C}$ of polynomial-sized circuits. If there exists a public-key (resp., private-key) bounded-key FE scheme for $\mathcal{C}$ then there exists a public-key (resp., private-key) bounded-key FE scheme for $\mathcal{C}$ that additionally satisfies linear complexity property (Definition 1).*

Remark 4. *Our transformation does not place any restrictions on $\mathcal{C}$. In particular, our transformation works for identity-based encryption schemes, attribute-based encryption schemes, and so on.*

The above theorem is restated as Theorem 4 in Section 4.

Bounded key FE for $\mathsf{P/Poly}$. We prove the following theorem in Section 5.

Theorem 2. *Assuming the existence of public-key encryption (resp., one-way functions), there exists a public-key (resp., private-key) bounded-key functional encryption scheme for $\mathsf{P/Poly}$.*

We prove the above theorem by first defining a client-server framework and then we construct a bounded-key FE from a client-server protocol. Finally, we instantiate client-server protocols from one-way functions.

The above theorem is restated as Theorem 5 in Section 5.

Bounded key FE for $\mathsf{P/Poly}$ satisfying Linear Complexity. By combining the above two theorems, we achieve our main result.

Theorem 3 (Main Theorem). *Assuming the existence of public-key encryption (resp., one-way functions), there exists a public-key (resp., private-key) bounded-key functional encryption scheme satisfying linear complexity property for $\mathsf{P/Poly}$.*

Our construction of functional encryption scheme in the above theorem makes only black box use of public-key encryption (or one-way functions).

4 Achieving Linear Complexity Generically

We show how to generically achieve linear complexity for any bounded-key FE scheme. In particular, we prove the following:

Theorem 4. *If there exists a bounded-key FE scheme, denoted by* bfe*, for* $\mathcal{C}$ *then there exists a bounded-key FE scheme, denoted by* BFE*, for* $\mathcal{C}$ *that additionally satisfies linear complexity property (Definition 1). Moreover, the following holds:*

- *If* bfe *is adaptively secure (resp., selectively secure) then* BFE *is adaptively secure (resp., selectively secure).*
- *If* bfe *is a public-key (resp., private key) scheme then* BFE *is a public-key (resp., private key) scheme.*
- *If* bfe *is simulation secure (resp., IND-secure) then* BFE *is simulation secure (resp., IND-secure).*

Proof. We focus on the case when bfe is adaptively secure, public-key and simulation secure. Our construction easily extends to the other cases as well.

We describe BFE below.

- $\underline{\mathsf{Setup}(1^\lambda, 1^Q, 1^s)}$: On input security parameter λ, query bound Q, maximum circuit size s for which functional keys are issued, generate $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{bfe.Setup}(1^\lambda, 1^q, 1^s)$ for every $i \in [Q]$, where $q = \lambda$ (in fact, q can even be set to be poly-logarithmic in the security parameter). Output the following:

$$\mathsf{MSK} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_Q)\,,\ \mathsf{MPK} = \big(\mathsf{mpk}_1, \ldots, \mathsf{mpk}_Q\big)$$

- $\underline{\mathsf{KeyGen}(\mathsf{msk}, C)}$: On input master secret key msk, circuit $C \in \mathcal{C}$,
  - Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$.
  - Generate $\mathsf{sk}_C \xleftarrow{\$} \mathsf{bfe.KeyGen}(\mathsf{bfe.msk}_{\boldsymbol{u}}, C)$.

  Output $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$.

- $\underline{\mathsf{Enc}(\mathsf{MPK}, x)}$: On input master public key MPK, input x, generate $\mathsf{ct}_i \leftarrow \mathsf{bfe.Enc}(\mathsf{mpk}_i, x)$ for every $i \in [Q]$. Output $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$.

- $\underline{\mathsf{Dec}(\mathsf{SK}_C, \mathsf{CT})}$: On input functional key $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$, ciphertext $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$, compute $\mathsf{bfe.Dec}(\mathsf{sk}_C, \mathsf{ct}_{\boldsymbol{u}})$. Output the result.

The correctness of BFE follows directly from the correctness of bfe. We analyze the efficiency of the above scheme next.

Suppose the time taken to generate a bfe ciphertext of message x is $\mathrm{poly}(\lambda, s)$ then the time taken to generate a BFE ciphertext of message x is $Q \cdot \mathrm{poly}(\lambda, s)$. Similarly, if the time taken to generate a bfe functional key of C is $\mathrm{poly}(\lambda, s)$, where s is the size of C, then the time taken to generate a BFE functional key of f is $Q \cdot \mathrm{poly}(\lambda, s)$. Thus, the resulting scheme BFE satisfies linear complexity property.

The only property left to be proved is the security property, which we prove next.

Security. Let sim be the stateful simulator of the bfe scheme. Since we invoke bfe scheme Q times in the scheme, we consider Q instantiations of the stateful simulator, denoted by $\mathsf{sim}_1, \ldots, \mathsf{sim}_Q$. We construct a simulator SIM associated with the BFE scheme. We denote the PPT adversary to be $\mathcal{A}$.

The simulator SIM proceeds as follows:

1. It receives the query bound Q and the maximum circuit size s from $\mathcal{A}$.

2. For every $i \in [Q]$, execute $\mathsf{sim}_i(1^\lambda, 1^Q, 1^s)$ to obtain the i^{th} master public key mpk_i. Set $\mathsf{MPK} = (\mathsf{mpk}_1, \ldots, \mathsf{mpk}_Q)$. Send MPK to $\mathcal{A}$.

3. Initialize the sets $\mathsf{qset}_i = \emptyset$, for every $i \in [Q]$. For every circuit query C made by $\mathcal{A}$, do the following:

   Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$ and then generate $\mathsf{sk}_C \leftarrow \mathsf{sim}_{\boldsymbol{u}}(C)$. Add C to $\mathsf{qset}_{\boldsymbol{u}}$. If $|\mathsf{qset}_{\boldsymbol{u}}| > q$ then output $\perp$. Otherwise, send $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$.

   $\mathcal{A}$ finally outputs the challenge message x.

4. For every $i \in [Q]$, construct the set $\mathcal{V}_i$ as follows: for every $C \in \mathsf{qset}_i$, include $(C, C(x))$ in $\mathcal{V}_i$. For every $i \in [Q]$, compute $\mathsf{sim}_i(1^{|x|}, \mathcal{V}_i)$ to obtain ct_i. Set $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$. Send CT to $\mathcal{A}$.

5. In the next phase, $\mathcal{A}$ makes circuit queries. For every circuit query C made by the adversary, do the following:

   Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$ and then generate $\mathsf{sk}_C \leftarrow \mathsf{sim}_{\boldsymbol{u}}(C, C(x))$. Add C to $\mathsf{qset}_{\boldsymbol{u}}$. If $|\mathsf{qset}_{\boldsymbol{u}}| > q$ then output $\perp$. Otherwise, send $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$.

Consider the following hybrids. The changes are marked in red.

$\underline{\mathbf{Hyb}_1}$: This corresponds to the real experiment. For completeness, we describe the real experiment here.

1. The challenger Ch receives the query bound Q and the maximum circuit size s from $\mathcal{A}$.

2. Execute $\mathsf{bfe.Setup}(1^\lambda, 1^Q, 1^s)$ for Q times to obtain $\{(\mathsf{msk}_i, \mathsf{mpk}_i)\}_{i \in [Q]}$. Set $\mathsf{MPK} = (\mathsf{mpk}_1, \ldots, \mathsf{mpk}_Q)$. Send MPK to $\mathcal{A}$.

3. Initialize the sets $\mathsf{qset}_i = \emptyset$, for every $i \in [Q]$. For every circuit query C made by $\mathcal{A}$, Ch does the following:

   Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$ and then generate $\mathsf{sk}_C \leftarrow \mathsf{bfe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, C)$. Add C to $\mathsf{qset}_{\boldsymbol{u}}$. Send $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$ to $\mathcal{A}$.

   $\mathcal{A}$ finally outputs the challenge message x.

4. For every $i \in [Q]$, generate $\mathsf{ct}_i \leftarrow \mathsf{bfe.Enc}(\mathsf{mpk}_i, x)$. Set $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$. Send CT to $\mathcal{A}$.

5. In the next phase, $\mathcal{A}$ makes circuit queries. For every circuit query C made by $\mathcal{A}$, do the following:

   Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$ and then generate $\mathsf{sk}_C \leftarrow \mathsf{bfe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, C)$. Add C to $\mathsf{qset}_{\boldsymbol{u}}$. Send $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$ to $\mathcal{A}$.

6. Let b be the output of $\mathcal{A}$. If $\left|\bigcup_{i=1}^{Q} \mathsf{qset}_i\right| > Q$, output $\perp$. Otherwise, output b.

Remark 5. *Note that the experiment described above is phrased differently from the real experiment of the bounded-key FE scheme. In the real experiment, the challenger only keeps track of the total number of* BFE *circuit queries made by the adversary but in* $\mathbf{Hyb}_1$*, the challenger keeps track of the set of* bfe *functional keys issued per index. Since ultimately the challenger only aborts if the size of the union of all these sets exceeds* Q*, the output distribution of* $\mathbf{Hyb}_1$ *is the same as the output distribution of the real experiment.*

$\underline{\mathbf{Hyb}_2}$: This hybrid is the same as the previous hybrid except that the real experiment outputs $\perp$ if there exists an index $\boldsymbol{u} \in [Q]$ such that $|\mathsf{qset}_{\boldsymbol{u}}| > q$.

In particular, we make the following changes to bullets 3 and 5 in the experiment described in $\mathbf{Hyb}_1$.

3. Initialize the sets $\mathsf{qset}_i = \emptyset$, for every $i \in [Q]$. For every circuit query C made by $\mathcal{A}$, Ch does the following:

   Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$ and then generate $\mathsf{sk}_C \leftarrow \mathsf{bfe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, C)$. Add C to $\mathsf{qset}_{\boldsymbol{u}}$. If $|\mathsf{qset}_{\boldsymbol{u}}| > q$ then output $\perp$. Otherwise, send $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$ to $\mathcal{A}$.

   $\mathcal{A}$ finally outputs the challenge message x.

5. In the next phase, $\mathcal{A}$ makes circuit queries. For every circuit query C made by $\mathcal{A}$, do the following:

   Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$ and then generate $\mathsf{sk}_C \leftarrow \mathsf{bfe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, C)$. Add C to $\mathsf{qset}_{\boldsymbol{u}}$. If $|\mathsf{qset}_{\boldsymbol{u}}| > q$ then output $\perp$. Otherwise, send $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$ to $\mathcal{A}$.

Claim 1. *The statistical distance between the output distributions of* $\mathbf{Hyb}_1$ *and* $\mathbf{Hyb}_2$ *is at most* $Q \cdot e^{-\frac{(q-1)^2}{1+q}}$ *and thus, negligible in* λ.

Proof. Define $\mathbf{X}_{\boldsymbol{u},j}$, for every $\boldsymbol{u} \in [Q], j \in [Q]$, to be a random variable such that $\mathbf{X}_{\boldsymbol{u},j} = 1$ if in the j^{th} circuit query C made by the adversary, the challenger responds with $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$; that is, the challenger responds with the functional key corresponding to the $\boldsymbol{u}^{th}$ instantiation of bfe. Let $\mathbf{X}_{\boldsymbol{u}} = \sum_{j=1}^{Q} \mathbf{X}_{\boldsymbol{u},j}$.

Note that $\Pr[\mathbf{X}_{\boldsymbol{u},j} = 1] = \frac{1}{Q}$. By linearity of expectation, $\mathbb{E}[\mathbf{X}_{\boldsymbol{u}}] = 1$.

By Chernoff bound, we have the following: for every $\boldsymbol{u} \in Q$,

$$\begin{aligned} \Pr[\mathbf{X}_{\boldsymbol{u}} > q] &= \Pr[\mathbf{X}_{\boldsymbol{u}} > q \cdot \mathbb{E}[\mathbf{X}_{\boldsymbol{u}}]] \\ &\leq \frac{1}{e^{\frac{(q-1)^2}{2+(q-1)} \cdot \mathbb{E}[\mathbf{X}_{\boldsymbol{u}}]}} \end{aligned}$$

Thus for any fixed $\boldsymbol{u} \in [Q]$, the probability that the number of bfe functional keys per index $\boldsymbol{u}$ issued by the challenger is greater than q is at most $e^{-\frac{(q-1)^2}{1+q}}$. By union bound, the probability that there exists an index $\boldsymbol{u}$ such that the challenger issues more than q functional keys with respect to $\boldsymbol{u}$ is at most $Q \cdot e^{-\frac{(q-1)^2}{1+q}}$. $\square$

Next, we consider a sequence of intermediate hybrids.

$\mathbf{Hyb}_{3.\mathbf{j}}$, for all $\mathbf{j} \in [Q]$: In this intermediate hybrid, the first $\boldsymbol{u}$ instantiations, with $\boldsymbol{u} < \mathbf{j}$ are simulated. The rest of the instantiations are honestly computed.

We consider $\mathbf{j}$ instantiations of the stateful simulator, denoted by $\mathsf{sim}_1, \ldots, \mathsf{sim}_{\mathbf{j}}$. We describe the hybrid experiment below.

1. The challenger Ch receives the query bound Q and the maximum circuit size s from $\mathcal{A}$.
2. For $i < \mathbf{j}$, execute $\mathsf{sim}_i(1^\lambda, 1^Q, 1^s)$ to obtain the i^{th} master public key mpk_i. For $i \geq \mathbf{j}$, execute $\mathsf{bfe.Setup}(1^\lambda, 1^Q, 1^s)$ to obtain $(\mathsf{msk}_i, \mathsf{mpk}_i)$. Set $\mathsf{MPK} = (\mathsf{mpk}_1, \ldots, \mathsf{mpk}_Q)$. Send MPK to $\mathcal{A}$.
3. Initialize the sets $\mathsf{qset}_i = \emptyset$, for every $i \in [Q]$. For every circuit query C made by $\mathcal{A}$, Ch does the following:

   Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$ and then generate sk_C as follows:

   - If $\boldsymbol{u} < \mathbf{j}$, generate $\mathsf{sk}_C \leftarrow \mathsf{sim}_{\boldsymbol{u}}(C)$.
   - If $\boldsymbol{u} \geq \mathbf{j}$, generate $\mathsf{sk}_C \leftarrow \mathsf{bfe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, C)$.

   Add C to $\mathsf{qset}_{\boldsymbol{u}}$. If $|\mathsf{qset}_{\boldsymbol{u}}| > q$ then output $\perp$. Otherwise, send $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$ to $\mathcal{A}$.

   $\mathcal{A}$ finally outputs the challenge message x.
4. For every $i \in [Q]$, construct the set $\mathcal{V}_i$ as follows: for every $C \in \mathsf{qset}_i$, include $(C, C(x))$ in $\mathcal{V}_i$. Compute CT as follows:

   - If $i < \mathbf{j}$, compute $\mathsf{sim}_i(1^{|x|}, \mathcal{V}_i)$ to obtain ct_i.
   - If $i \geq \mathbf{j}$, compute $\mathsf{ct}_i \leftarrow \mathsf{bfe.Enc}(\mathsf{mpk}_i, x)$.

   Set $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$. Send CT to $\mathcal{A}$.
5. In the next phase, $\mathcal{A}$ makes circuit queries. For every circuit query C made by $\mathcal{A}$, do the following:

   Sample $\boldsymbol{u} \xleftarrow{\$} [Q]$ and then generate sk_C as follows:

   - If $\boldsymbol{u} < \mathbf{j}$, generate $\mathsf{sk}_C \leftarrow \mathsf{sim}_{\boldsymbol{u}}(C, C(x))$.
   - If $\boldsymbol{u} \geq \mathbf{j}$, generate $\mathsf{sk}_C \leftarrow \mathsf{bfe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, C)$.

   Add C to $\mathsf{qset}_{\boldsymbol{u}}$. If $|\mathsf{qset}_{\boldsymbol{u}}| > q$ then output $\perp$. Otherwise, send $\mathsf{SK}_C = (\boldsymbol{u}, \mathsf{sk}_C)$ to $\mathcal{A}$.
6. Let b be the output of $\mathcal{A}$. If $\left|\bigcup_{i=1}^{Q} \mathsf{qset}_i\right| > Q$, output $\perp$. Otherwise, output b.

The following two claims are immediate.

Claim 2. *The output distributions of* $\mathbf{Hyb}_2$ *and* $\mathbf{Hyb}_{3.1}$ *are identically distributed.*

Claim 3. *For every* $\mathbf{j} \in [Q-1]$, *the security of* bfe *implies that the output distributions of* $\mathbf{Hyb}_{3.\mathbf{j}}$ *and* $\mathbf{Hyb}_{3.\mathbf{j}+1}$ *are computationally indistinguishable.*

$\underline{\mathbf{Hyb}_4}$: This corresponds to the simulated experiment.

The proof of the following claim is immediate.

Claim 4. *The security of* bfe *implies that the output distributions of* $\mathbf{Hyb}_{3.Q}$ *and* $\mathbf{Hyb}_4$ *are computationally indistinguishable.*

$\square$

5 Construction of Bounded-Key FE for $\mathsf{P/Poly}$

We construct a bounded-key FE scheme for $\mathsf{P/Poly}$ as follows:

- First we define a client-server framework and show how to construct a bounded-key FE for $\mathsf{P/Poly}$ from a protocol in this client-server framework.
- Next, we show how to construct a protocol in the client-server framework from one-way functions.

5.1 Client-Server Framework

The client-server framework consists of a single client and $N = N(\lambda, Q)$ servers, where λ is the security parameter. It is additionally parameterized by $n = n(\lambda, Q)$ and $t = t(\lambda, Q)$. The framework consists of the following two phases:

- **Offline Phase**: In this phase, the client takes as input the number of sessions Q, size of the circuit delegated s, input x and executes a PPT algorithm InpEnc that outputs correlated input encodings $(\widehat{x}^1, \ldots, \widehat{x}^N)$. It sends the encoding $\widehat{x}^{\boldsymbol{u}}$ to the $\boldsymbol{u}^{th}$ server.
- **Online Phase**: This phase is executed for Q sessions. In each session, the client delegates the computation of a circuit C on x to the servers. This is done in the following steps:
  - **Client Delegation**: This is performed by the client computing a PPT algorithm CktEnc on input $(1^\lambda, 1^Q, 1^s, C)$ to obtain $(\widehat{C}^1, \ldots, \widehat{C}^N)$. It sends the circuit encoding $\widehat{C}^{\boldsymbol{u}}$ to the $\boldsymbol{u}^{th}$ server. Note that CktEnc is executed independently of the offline phase and in particular, does not depend on the randomness used in the offline phase[2].
  - **Local Computation by Servers**: Upon receiving the circuit encodings from the client, a subset $\mathbf{S}$ of servers come online and the $\boldsymbol{u}^{th}$ server in this set $\mathbf{S}$ computes $\mathsf{Local}(\widehat{C}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}})$ to obtain the $\boldsymbol{u}^{th}$ output encoding $\widehat{y}^{\boldsymbol{u}}$.
  - **Decoding**: Finally, the output is recovered by computing a PPT algorithm Decode on $\left(\{\widehat{y}^{\boldsymbol{u}}\}_{\boldsymbol{u}\in\mathbf{S}}, \mathbf{S}\right)$.

We describe the properties below. We start with correctness.

[2]We could define a notion where CktEnc takes as input the randomness of the offline phase. It is however not clear how to build FE from such a notion.

Correctness. A protocol Π in the client-server framework is said to be correct if the following holds:

- Suppose the client computes encodings of input x by computing $(\widehat{x}^1, \ldots, \widehat{x}^N) \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, 1^s, x)$.
- In the online phase, let C be the circuit that the client wants to delegate. The client computes $(\widehat{C}^1, \ldots, \widehat{C}^N) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C)$ and distributes the circuit encodings to all the servers. A subset of servers $\mathbf{S} \subseteq [N]$, of size n, then locally compute on the circuit encodings. That is, for every $\boldsymbol{u} \in \mathbf{S}$, the $\boldsymbol{u}^{th}$ server computes $\widehat{y}^{\boldsymbol{u}} = \mathsf{Local}(\mathsf{gc}, \widehat{x}^{\boldsymbol{u}})$. Finally, the output can be recovered by computing $\mathsf{Decode}(\{\widehat{y}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathbf{S}}, \mathbf{S})$ to obtain y.

We require that $y = C(x)$.

5.1.1 Security

We allow the adversary to be able to corrupt a subset of servers. Once the server is corrupted, the entire state of the server is leaked to the adversary. The adversary, however, is not allowed to corrupt the client. In every session, since n servers can recover the output, the number of servers that can be corrupted has to be less than n[3].

Informally, we require the following guarantee: even if the adversary can corrupt a subset of servers, he cannot learn anything beyond the outputs of the computation $(C_1(x), \ldots, C_Q(x))$ in every session, where $C_1, \ldots, C_Q$ are the circuits delegated by the client. However, the circuits $(C_1, \ldots, C_Q)$ are not hidden from the adversary. Since our end goal is to build FE for $\mathsf{P/Poly}$, we need to suitably define the security property that would enable us to prove the security of FE. Towards this, we incorporate the following in the security definition of the client-server framework:

- We not only allow the adversary to choose the servers to corrupt but also allow it to decide the subsets of servers $\mathbf{S}_1, \ldots, \mathbf{S}_Q$ participating in the Q sessions.
- In every session, the adversary is provided all the N circuit encodings. Moreover, the outputs of the local computation of all the servers, including the honest servers, are visible to the adversary.

To define the security notion formally, we first state the following experiments. The first experiment Expt_0 is parameterized by a PPT adversary $\mathcal{A}$ and PPT challenger Ch and the second experiment Expt_1 is parameterized by $\mathcal{A}$ and PPT stateful simulator $\mathsf{Sim}_{\mathsf{csf}}$.

$\underline{\mathsf{Expt}_0^{\mathcal{A},\mathsf{Ch}}(1^\lambda)}$:

- $\mathcal{A}$ outputs the query bound Q, maximum circuit size s, total number of parties N, number of parties n participating in any session, threshold t, corruption set $\mathcal{S}_{\mathsf{corr}} \subseteq [N]$ and the input x. If $|\mathcal{S}_{\mathsf{corr}}| > t$ then the experiment aborts. It also outputs the sets $\mathbf{S}_1, \ldots, \mathbf{S}_Q \subseteq [N]$ such that $|\mathbf{S}_i| = n$, where $\mathbf{S}_i$ is the set of parties participating in the i^{th} session.

[3]If n or more servers can be corrupted then the corrupted set can recover $C(x)$ for any circuit C: this is because the corrupted servers can execute CktEnc on input C, run the local computation procedure and then decode their outputs. Thus, such a notion would imply program obfuscation.

- **Circuit Queries**: $\mathcal{A}$ is allowed to make a total of Q circuit queries. First, it makes $Q_1 \leq Q$ adaptive[4] circuit queries $C_1, \ldots, C_{Q_1}$.

  For the i^{th} circuit query C_i, Ch computes $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C_i)$ and sends $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right)$.

- **Challenge Input Query**: $\mathcal{A}$ submits the input x. Ch generates $\mathsf{InpEnc}(1^\lambda, 1^Q, 1^s, x)$ to obtain $(\widehat{x}^1, \ldots, \widehat{x}^N)$.

  Ch sends $\left(\{\widehat{x}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}}, \left\{\mathsf{Local}\left(\widehat{C_i}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}}\right)\right\}_{i \in [Q_1], \boldsymbol{u} \in \mathbf{S}_i}\right)$. That is, the challenger sends the input encodings of the corrupted set of servers along with the outputs of Local on the circuit encodings received so far.

- **Circuit Queries**: $\mathcal{A}$ then makes $Q_2 = Q - Q_1$ adaptive circuit queries $C_{Q_1+1}, \ldots, C_Q$.

  Ch computes $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C_i)$ and sends
  $\left\{\left\{\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right), \; \mathsf{Local}\left(\widehat{C_i}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}}\right)\right\}_{i \in \{Q_1+1, \ldots, Q\}, \boldsymbol{u} \in [\mathbf{S}_i]}\right\}$.

- $\mathcal{A}$ outputs a bit b. The output of the experiment is b.

$\underline{\mathsf{Expt}_1^{\mathcal{A}, \mathsf{Sim}_{\mathsf{csf}}}(1^\lambda)}$:

- $\mathcal{A}$ outputs the query bound Q, maximum circuit size s, total number of parties N, number of parties n participating in any session, threshold t, corruption set $\mathcal{S}_{\mathsf{corr}} \subseteq [N]$ and the input x. If $|\mathcal{S}_{\mathsf{corr}}| > t$ then the experiment aborts. It also outputs the sets $\mathbf{S}_1, \ldots, \mathbf{S}_Q \subseteq [N]$ such that $|\mathbf{S}_i| = n$, where $\mathbf{S}_i$ is the set of parties participating in the i^{th} session.

- **Circuit Queries**: $\mathcal{A}$ makes a total of Q adaptive queries. First it makes $Q_1 \leq Q$ adaptive circuit queries. For the i^{th} circuit query C_i, the simulator computes $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{Sim}_{\mathsf{csf}}(C_i)$ and sends $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right)$.

- **Challenge Input Query**: $\mathcal{A}$ submits the input x. Construct $\mathcal{V}$ as follows: $\mathcal{V} = \{C_i, C_i(x) : i \in [Q_1]\}$.

  $\mathsf{Sim}_{\mathsf{csf}}$ on input $(1^{|x|}, \mathcal{S}_{\mathsf{corr}}, \mathcal{V})$ (and in particular, it does not get x as input) outputs the simulated encodings $(\{\widehat{x}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}})$ and the encodings of outputs $\{\widehat{y}_i^{\boldsymbol{u}}\}_{i \in [Q_1], \boldsymbol{u} \in \mathbf{S}_i}$ such that $\widehat{y}_i^{\boldsymbol{u}} = \mathsf{Local}\left(\widehat{C_i}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}}\right)$ for every $\boldsymbol{u} \in \mathbf{S}_i \cap \mathcal{S}_{\mathsf{corr}}$.

- **Circuit Queries**: $\mathcal{A}$ then makes $Q_2 = Q - Q_1$ adaptive circuit queries $C_{Q_1+1}, \ldots, C_Q$. The simulator $\mathsf{Sim}_{\mathsf{csf}}$ on input $(i, C_i, C_i(x))$, for $i \in \{Q_1+1, \ldots, Q\}$, sends $\left(\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right), \; \widehat{y}_i^{\boldsymbol{u}}\right)$.

- $\mathcal{A}$ outputs a bit b. The output of the experiment is b.

We formally define the security property below.

[4]By adaptive, we mean that the adversary can decide each circuit query as a function of all the previous circuit queries.

Definition 3 (Security)**.** *A protocol* Π *is secure if for every PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* $\mathsf{Sim}_{\mathsf{csf}}$ *such that the following holds:*

$$\left|\Pr[0 \leftarrow \mathsf{Expt}_0^{\mathcal{A},\mathsf{Ch}}(1^\lambda)] - \Pr[0 \leftarrow \mathsf{Expt}_1^{\mathcal{A},\mathsf{Sim}_{\mathsf{csf}}}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda),$$

for some negligible function negl.

5.2 Construction of Protocols in Client-Server Framework

Towards constructing a protocol in client-server framework, we first define the concept of correlated garbling and associated with this are two PPT algorithms $\mathsf{CorrGarb}$ and $\mathsf{CorrEval}$. We describe these two algorithms and prove properties associated with these algorithms.

5.2.1 Correlated Garbling

Consider the "point-and-permute" distributed garbling scheme proposed in the work of Beaver, Micali and Rogaway [8]. We use this scheme as a starting point to design $\mathsf{CorrGarb}$ and $\mathsf{CorrEval}$.

Starting Point: Point-and-Permute Garbling Scheme [8]. We recall the scheme of point-and-permute garbling scheme. As in a standard garbling scheme, the scheme of [8] is associated with garbling and decoding algorithms. The garbling algorithm takes as input a circuit C and outputs a garbling of C along with ℓ pairs of wire keys $(\boldsymbol{K}_{w_1,0}, \boldsymbol{K}_{w_1,1}), \ldots, (\boldsymbol{K}_{w_\ell,0}, \boldsymbol{K}_{w_\ell,1})$, where ℓ is the input length and $w_1, \ldots, w_\ell$ are the input wires of C. The decoding algorithm takes as input the garbling of C along with $(\boldsymbol{K}_{w_1,x_1}, \ldots, \boldsymbol{K}_{w_\ell,x_\ell})$ and outputs $C(x)$.

However, unlike a standard garbling scheme, there are two main differences:

- The label associated with wire w is of the form $\boldsymbol{K}_{w,b} = (\boldsymbol{k}_{w,b\oplus r_w}, b \oplus r_w)$, where $\boldsymbol{k}_{w,0}, \boldsymbol{k}_{w,1}$ are random strings and r_w is a random bit associated with wire w. Moreover, interpret $\boldsymbol{k}_{w,0}$ (resp., $\boldsymbol{k}_{w,1}$) as a concatenation of two strings $\boldsymbol{k}^0_{w,0}$ (resp., $\boldsymbol{k}^0_{w,1}$) and $\boldsymbol{k}^1_{w,0}$ (resp., $\boldsymbol{k}^1_{w,1}$).
- The garbled gate associated with a gate G consist of four entries, indexed by $00, 01, 10$ and 11. Let w_1, w_2 be the input wires of G and let w_3 and w_4 be its output wires. The $(b_1, b_2)^{th}$ entry, for $b_1 \in \{0,1\}, b_2 \in \{0,1\}$, is of the following form:

$$\begin{aligned}\boldsymbol{k}^{b_2}_{w_1,b_1} \oplus \boldsymbol{k}^{b_1}_{w_2,b_2} \quad \oplus \quad & \big((\boldsymbol{k}_{w_3,G(b_1\oplus r_{w_1},b_2\oplus r_{w_2})\oplus r_{w_3}}) \circ (G(b_1 \oplus r_{w_1}, b_2 \oplus r_{w_2}) \oplus r_{w_3}) \circ \\ & (\boldsymbol{k}_{w_4,G(b_1\oplus r_{w_1},b_2\oplus r_{w_2})\oplus r_{w_4}}) \circ (G(b_1 \oplus r_{w_1}, b_2 \oplus r_{w_2}) \oplus r_{w_4})\big)\end{aligned}$$

  Recall that '$\circ$' is a concatenation operation.

For the output wires, provide a translation table that maps the output wire labels to bits 0 and 1. This translation table will be part of the garbled circuit GC.

Note that the above scheme only works for log-depth circuits since the wire labels for the input wires of a gate are required to be twice as large as the wire labels for the output wires of a gate. This scheme can be adapted to achieve a scheme for $\mathsf{P/Poly}$ as follows: the key $\boldsymbol{k}_{w,b}$ is generated as an output of a PRG seed $\mathbf{sd}_{w,b}$. Concretely, we consider a pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \rightarrow$

$\{0,1\}^{2\lambda+2}$ and generate $\mathsf{PRG}(\mathbf{sd}_{w,b}) = \boldsymbol{k}_{w,b}$. The $(b_1, b_2)^{th}$ entry, for $b_1 \in \{0,1\}, b_2 \in \{0,1\}$, is instead generated as:

$$\begin{aligned}\boldsymbol{k}^{b_2}_{w_1,b_1} \oplus \boldsymbol{k}^{b_1}_{w_2,b_2} \quad \oplus \quad & \big((\mathbf{sd}_{w_3,G(b_1\oplus r_{w_1},b_2\oplus r_{w_2})\oplus r_{w_3}}) \circ (G(b_1 \oplus r_{w_1}, b_2 \oplus r_{w_2}) \oplus r_{w_3}) \circ \\ & (\mathbf{sd}_{w_4,G(b_1\oplus r_{w_1},b_2\oplus r_{w_2})\oplus r_{w_4}}) \circ (G(b_1 \oplus r_{w_1}, b_2 \oplus r_{w_2}) \oplus r_{w_4})\big)\end{aligned}$$

We use this scheme, adapted for $\mathsf{P/Poly}$, to design auxiliary algorithms, described next.

Correlated garbling algorithms $\mathsf{CorrGarb}$ and $\mathsf{CorrEval}$. The algorithms will be parameterized by $T = \Theta(\mathsf{v}Q^2)$ and $\mathsf{v} = \Theta(\lambda)$, where Q is the query bound.

Consider a pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\mathsf{v}\lambda+2}$.

- $\underline{\mathsf{CorrGarb}\left(\Delta, C, x, \{\mathbf{sd}^j_{w,b}, \boldsymbol{k}^j_{w,b}\}_{\substack{j\in[T],w\in\mathcal{W},\\ b\in\{0,1\}}}, \{\mathbf{r}^j_w\}_{j\in[T],w\in\mathcal{W}}\right)}$: On input, $\Delta \subseteq [T]$ of size v, circuit $C : \{0,1\}^L \to \{0,1\}$, $T+|\mathcal{W}|+2$ PRG seeds $\{\mathbf{sd}^j_{w,b}\}$ and its outputs $\{\boldsymbol{k}^j_{w,b} = \mathsf{PRG}(\mathbf{sd}^j_{w,b})\}$, where $\mathcal{W}$ is the set of wires in $\mathcal{U}(\cdot,\cdot)$ with $\mathcal{U}$ being a universal circuit that takes as input circuit of length $|C|$ and input of length $|x|$, $T + |\mathcal{W}|$ wire masks, do the following:
  - For every wire $w \in \mathcal{W}$, bit $b \in \{0,1\}$, set the wire key to be $\boldsymbol{k}_{w,b} = \bigoplus_{j\in\Delta} \boldsymbol{k}^j_{w,b}$. Also, set the wire label to be $\mathbf{r}_w = \bigoplus_{j\in\Delta} \mathbf{r}^j_w$.
  - For any gate G in $\mathcal{U}(\cdot,\cdot)$, generate the garbled table for G as follows: let the input wires of G be w_1, w_2 and the output wires of G be w_3, w_4. For every $b_1, b_2 \in \{0,1\}$,
    * Set $\mathbf{sd}_{w_3,G(b_1,b_2)\oplus\mathbf{r}_{w_3}}$ to be $\left(\mathbf{sd}^{j_1}_{w_3,G(b_1,b_2)\oplus\mathbf{r}_{w_3}} \circ \cdots \circ \mathbf{sd}^{j_\mathsf{v}}_{w_3,G(b_1,b_2)\oplus\mathbf{r}_{w_3}}\right)$, where $\Delta = \{j_1, \ldots, j_\mathsf{v}\}$.
    * Similarly, set $\mathbf{sd}_{w_4,G(b_1,b_2)\oplus\mathbf{r}_{w_4}} = \mathbf{sd}^{j_1}_{w_4,G(b_1,b_2)\oplus\mathbf{r}_{w_4}} \circ \cdots \circ \mathbf{sd}^{j_\mathsf{v}}_{w_4,G(b_1,b_2)\oplus\mathbf{r}_{w_4}}$.
    * The $(b_1, b_2)^{th}$ entry in the garbled table of G is set to be:

    $$\begin{aligned}\boldsymbol{k}^{b_2}_{w_1,b_1} \oplus \boldsymbol{k}^{b_1}_{w_2,b_2} \quad \oplus \quad & (\mathbf{sd}_{w_3,G(b_1\oplus\mathbf{r}_{w_1},b_2\oplus\mathbf{r}_{w_2})\oplus\mathbf{r}_{w_3}} \circ G(b_1 \oplus \mathbf{r}_{w_1}, b_2 \oplus \mathbf{r}_{w_2}) \oplus \mathbf{r}_{w_3} \circ \\ & \mathbf{sd}_{w_4,G(b_1\oplus\mathbf{r}_{w_1},b_2\oplus\mathbf{r}_{w_2})\oplus\mathbf{r}_{w_4}} \circ G(b_1 \oplus \mathbf{r}_{w_1}, b_2 \oplus \mathbf{r}_{w_2}) \oplus \mathbf{r}_{w_4}),\end{aligned}$$

    where $\boldsymbol{k}_{w_1,b_1} = (\boldsymbol{k}^0_{w_1,b_1} \circ \boldsymbol{k}^1_{w_1,b_1})$ and $\boldsymbol{k}^1_{w_2,b_2} = (\boldsymbol{k}^0_{w_2,b_2} \circ \boldsymbol{k}^1_{w_2,b_2})$.

  The translation table for the output wire is of the following form $(\boldsymbol{k}_{\mathsf{ow},0\oplus\mathbf{r}_{\mathsf{ow}}} \to 0, \boldsymbol{k}_{\mathsf{ow},1\oplus\mathbf{r}_{\mathsf{ow}}} \to 1)$, where ow is the output wire of C. Set the garbled circuit GC to consist of garbled tables for every gate in the circuit along with the translation table. Also generate the input wire keys corresponding to $\mathsf{inp} = (C \circ x)$.

  Output GC along with the following input wire labels:

  $$\mathbf{K} = \left((\mathbf{sd}_{\mathsf{iw}_1,\mathsf{inp}_1\oplus\mathbf{r}_{\mathsf{iw}_1}}; \mathsf{inp}_1 \oplus \mathbf{r}_{\mathsf{iw}_1}) \circ \cdots \circ (\mathbf{sd}_{\mathsf{iw}_L,\mathsf{inp}_L\oplus\mathbf{r}_{\mathsf{iw}_L}}; \mathsf{inp}_L \oplus \mathbf{r}_{\mathsf{iw}_L})\right),$$

  where (i) $\mathbf{sd}_{\mathsf{iw}_i,b} = \mathbf{sd}^{j_1}_{\mathsf{iw}_i,b} \circ \cdots \circ \mathbf{sd}^{j_v}_{\mathsf{iw}_i,b}$, (ii) $L = |C| + |x|$ and, (iii) $\mathsf{iw}_1, \ldots, \mathsf{iw}_L$ are the input wire labels of the universal circuit $\mathcal{U}(\cdot,\cdot)$.

- $\underline{\mathsf{CorrEval}\left(\mathsf{GC}, \mathbf{K}_{\mathsf{inp}}\right)}$: On input the garbled circuit GC, wire labels $\mathbf{K}_{\mathsf{inp}}$, for every gate G with input wires w_1, w_2 and output wires w_3, w_4, do the following: suppose $(\boldsymbol{k}_{w_1, b_1 \oplus \mathbf{r}_{w_1}}, b_1 \oplus \mathbf{r}_{w_1})$ and $\left(\boldsymbol{k}_{w_2, b_2 \oplus \mathbf{r}_{w_2}}, b_2 \oplus \mathbf{r}_{w_2}\right)$ be the wire labels respectively corresponding to the wires w_1 and w_2 obtained during the evaluation process. Using these keys and the $(b_1 \oplus \mathbf{r}_{w_1}, b_2 \oplus \mathbf{r}_{w_2})^{th}$ entry of the garbled table, we get the key $\mathbf{sd}_{w_3, G(b_1,b_2) \oplus \mathbf{r}_{w_3}} \circ G(b_1, b_2) \oplus \mathbf{r}_{w_3}$ for the w_3^{th} wire and the key $\mathbf{sd}_{w_4, G(b_1,b_2) \oplus \mathbf{r}_{w_4}} \circ G(b_1, b_2) \oplus \mathbf{r}_{w_4}$ for the w_4^{th} wire. Continuing this way, we finally obtain the labels associated with the output wire.

  Let the wire labels obtained corresponding to the output wires be $\boldsymbol{k}_{\mathsf{ow}_1, y_1 \oplus \mathbf{r}_{\mathsf{ow}_1}}, \ldots, \boldsymbol{k}_{\mathsf{ow}_{L'}, y_{L'} \oplus \mathbf{r}_{\mathsf{ow}_{L'}}}$. Using the translation table, output $y = (y_1 \circ \cdots \circ y_{L'})$.

Note that every output of $\mathsf{CorrGarb}(\cdot)$ can be represented by a cubic polynomial over $\mathbb{F}$, where $\mathbb{F}$ is any binary extension field (i.e, extension of $\mathbb{Z}_2$). The main difference between $\mathsf{CorrGarb}$ and the description of the garbling scheme of [8] is in the first step of $\mathsf{CorrGarb}$ and specifically in the generation of the wire keys $\{\boldsymbol{k}_{w,b}\}$ and labels $\{\mathbf{r}_w\}$. Observe that the generation of the wire keys $\{\boldsymbol{k}_{w,b}\}$ and labels $\{\mathbf{r}_w\}$ can be represented by linear polynomials over $\mathbb{F}$ (here we use the fact that $\mathbb{F}$ is an extension field of $\mathbb{Z}_2$). Moreover, the output of the garbling procedure in [8] can be represented by cubic polynomials over $\mathbb{F}$ (see [7]). Combining these two observations, we deduce that every output of $\mathsf{CorrGarb}(\cdot)$ can be represented by a cubic polynomial over $\mathbb{F}$.

Consider the following lemma.

Lemma 1 (Correctness of $(\mathsf{CorrGarb}, \mathsf{CorrEval})$)**.** *Consider a circuit C and an input x. Consider the following process:*

- *Let $\Delta \subseteq [T]$ of size v,*
- *For every wire $w \in \mathcal{W}$, index $j \in [T]$, bit $b \in \{0,1\}$, sample $\mathbf{sd}_{w,b}^{j} \xleftarrow{\$} \{0,1\}^{\lambda}$ and sample $\mathbf{r}_w^j \xleftarrow{\$} \{0,1\}$.*
- *For every wire $w \in \mathcal{W}$, index $j \in [T]$, bit $b \in \{0,1\}$, let $\mathbf{k}_{w,b}^{j} = \mathsf{PRG}\left(\mathbf{sd}_{w,b}^{j}\right)$.*
- *Let $(\mathsf{GC}, \mathbf{K}) \leftarrow \mathsf{CorrGarb}\left(\Delta, C, x, \left\{\mathbf{sd}_{w,b}^{j}, \mathbf{k}_{w,b}^{j}\right\}_{\substack{j \in [T], w \in \mathcal{W}, \\ b \in \{0,1\}}}, \left\{\mathbf{r}_w^j\right\}_{j \in [T], w \in \mathcal{W}}\right)$.*
- *Let $y \leftarrow \mathsf{CorrEval}\left(\mathsf{GC}, \mathbf{K}\right)$.*

Then, $y = C(x)$.

We omit the proof of the above lemma since it follows along the same lines as the correctness of point-and-permute distributed garbling scheme of [8].

We now state the security property guaranteed by $(\mathsf{CorrGarb}, \mathsf{CorrEval})$.

Security. Before we state the security property satisfied by $(\mathsf{CorrGarb}, \mathsf{CorrEval})$, we describe the following experiments; the first one is the real experiment and the second one is the simulated experiment associated with a PPT simulator $\mathsf{Sim}_{\mathsf{Corr}}$. Both the experiments are associated with the PPT adversary $\mathcal{A}$.

$\underline{\mathsf{CorrExpt}_0^{\mathcal{A},\mathsf{Ch}}(1^{\lambda})}$:

- Challenger Ch computes the following:
  - For $i \in [Q]$, Δ_i is a v-sized set sampled uniformly at random from $[T]$,
  - For every wire $w \in \mathcal{W}$, index $j \in [T]$, bit $b \in \{0,1\}$, sample $\mathbf{sd}^j_{w,b} \xleftarrow{\$} \{0,1\}^\lambda$ and sample $\mathbf{r}^j_w \xleftarrow{\$} \{0,1\}$.
  - For every wire $w \in \mathcal{W}$, index $j \in [T]$, bit $b \in \{0,1\}$, let $\boldsymbol{k}^j_{w,b} = \mathsf{PRG}\left(\mathbf{sd}^j_{w,b}\right)$.
- $\mathcal{A}$ submits its challenge input x. It then submits at most Q circuit queries $C_1, \ldots, C_Q$ adaptively.
- For the i^{th} circuit query C_i, Ch generates the following:

$$(\mathsf{GC}_i, \mathbf{K}_i) \leftarrow \mathsf{CorrGarb}\left(\Delta_i, C_i, x, \{\mathbf{sd}^j_{w,b}, \boldsymbol{k}^j_{w,b}\}_{\substack{j \in [T], w \in \mathcal{W},\\ b \in \{0,1\}}}, \{\mathbf{r}^j_w\}_{j \in [T], w \in \mathcal{W}}\right)$$

  It sends $\left(\{\mathsf{GC}_i, \mathbf{K}_i\}_{i \in [Q]}\right)$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs bit b. Output b.

$\underline{\mathsf{CorrExpt}_1^{\mathcal{A},\mathsf{Sim}_{\mathsf{Corr}}}}$:

- $\mathcal{A}$ submits its challenge input x. It then submits at most Q circuit queries $C_1, \ldots, C_Q$ adaptively.
- On input $(1^\lambda, C_i, C_i(x))$ (in particular, the simulator doesn't get x as input), $\mathsf{Sim}_{\mathsf{Corr}}$ generates $\left(\{\mathsf{GC}_i, \mathbf{K}_i\}_{i \in [Q]}\right)$. The result is sent to $\mathcal{A}$.
- $\mathcal{A}$ outputs bit b. Output b.

We now state the following lemma.

Lemma 2 (Correlated Garbling Lemma)**.** *For every PPT adversary $\mathcal{A}$, there exist a PPT simulator $\mathsf{Sim}_{\mathsf{Corr}}$ such that the following holds:*

$$\left|\Pr\left[0 \leftarrow \mathsf{CorrExpt}_0^{\mathcal{A},\mathsf{Ch}}(1^\lambda)\right] - \Pr\left[0 \leftarrow \mathsf{CorrExpt}_1^{\mathcal{A},\mathsf{Sim}_{\mathsf{Corr}}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda),$$

for some negligible function negl.

Proof. We first argue that with overwhelming probability, it holds that every set Δ_i contains a unique index that is not present in the other sets. We use this observation to argue that the garbled circuit GC_i, for every $i \in [Q]$, is secure. In more detail, we need to argue that for every wire w in GC_i, the PPT adversary can only obtain one wire key, say $\boldsymbol{k}_{w,b}$, during evaluation and in particular the other key $\boldsymbol{k}_{w,\bar{b}}$ is hidden.

Note that $\boldsymbol{k}_{w,\bar{b}}$ is obtained by xor-ing the outputs of many PRG seeds; that is, $\boldsymbol{k}_{w,\bar{b}} = \oplus_{j \in \Delta} \boldsymbol{k}^j_{w,\bar{b}}$ where $\boldsymbol{k}^j_{w,\bar{b}}$ is obtained by computing $\mathsf{PRG}(\mathbf{sd}^j_{w,\bar{b}})$. Moreover, some of $\{\mathbf{sd}^j_{w,\bar{b}}\}$ and thus, $\{\boldsymbol{k}^j_{w,\bar{b}}\}$ can be leaked during the execution of the other garbled circuits. Here, we use unique index of Δ_i, say j^*, to argue that the PRG seed $\mathbf{sd}^{j^*}_{w,\bar{b}}$ is information-theoretically hidden from the adversary. Assuming the security of pseudorandom generators, this further implies that the key $\{\boldsymbol{k}^j_{w,\bar{b}}\}$ is hidden from the adversary.

We describe the simulator $\mathsf{Sim}_{\mathsf{Corr}}$.

- For every $i \in [Q]$, sample a v-sized set $\Delta_i \subseteq [T]$ uniformly at random. Check if for every $i^* \in [Q]$, $\Delta_{i^*} \setminus \left(\cup_{\substack{i \in [Q] \\ i \neq i^*}} \Delta_i \right) \neq \emptyset$. That is, check that every Δ_{i^*} contains a unique index that is not present in the other sets. If this condition does not hold, output $\perp$.

- For every wire $w \in \mathcal{W}$, sample the PRG seeds as follows: for every $j \in [T]$, sample $\mathbf{sd}^j_{w,0} \xleftarrow{\$} \{0,1\}^{\lambda}$.

- $\mathcal{A}$ submits the input x. Then it adaptively submits the circuit queries $C_1, \ldots, C_Q$.

- For every $i \in [Q]$, the i^{th} simulated garbling GC_i of C_i is generated as follows: let $\Delta_i = \{j_1, \ldots, j_{\mathsf{v}}\}$.

  - For every wire $w \in \mathcal{W}$ and bit b, set $\mathbf{sd}_{w,b} = \mathbf{sd}^{j_1}_{w,b} \circ \cdots \circ \mathbf{sd}^{j_{\mathsf{v}}}_{w,b}$.
  - For every wire $w \in \mathcal{W}$, sample $\mathbf{r}_w \xleftarrow{\$} \{0,1\}$ uniformly at random.
  - Next, generate the input wire labels of GC_i. That is,

$$\mathbf{K}_i = \Big((\mathbf{sd}_{\mathsf{iw}_1, \mathbf{r}_{\mathsf{iw}_1}}; \mathbf{r}_{\mathsf{iw}_1}) \circ \cdots \circ (\mathbf{sd}_{\mathsf{iw}_L, \mathbf{r}_{\mathsf{iw}_L}}; \mathbf{r}_{\mathsf{iw}_L}) \Big)$$

  - For every gate G in $\mathcal{U}(\cdot,\cdot)$, where $\mathcal{U}(\cdot,\cdot)$ is the universal circuit that takes as input (C,x) and outputs $C(x)$, generate a garbled table for G as follows. Let w_1, w_2 be the input wires and w_3, w_4 be the output wires of C. Set the $(\mathbf{r}_{w_1}, \mathbf{r}_{w_2})^{th}$ entry in the garbled table to be the following:

$$\boldsymbol{k}^{\mathbf{r}_{w_2}}_{w_1,\mathbf{r}_{w_1}} \oplus \boldsymbol{k}^{\mathbf{r}_{w_1}}_{w_2,\mathbf{r}_{w_2}} \oplus \left(\mathbf{sd}_{w_3,\mathbf{r}_{w_3}} \circ \mathbf{r}_{w_3} \circ \mathbf{sd}_{w_4,\mathbf{r}_{w_4}} \circ \mathbf{r}_{w_4} \right),$$

    where $\bigoplus_{j \in \Delta_i} \mathsf{PRG}(\mathbf{sd}^j_{w_1,\mathbf{r}_{w_1}}) = \boldsymbol{k}^0_{w_1,\mathbf{r}_{w_1}} \circ \boldsymbol{k}^1_{w_1,\mathbf{r}_{w_1}}$ and $\bigoplus_{j \in \Delta_i} \mathsf{PRG}(\mathbf{sd}^j_{w_2,\mathbf{r}_{w_2}}) = \boldsymbol{k}^0_{w_2,\mathbf{r}_{w_2}} \circ \boldsymbol{k}^1_{w_2,\mathbf{r}_{w_2}}$. The other three entries in the garbled table are set to be random strings of length $2\mathsf{v}\lambda + 2$.

    Construct GC_i to consist of garbled gates for every gate in $\mathcal{U}(\cdot,\cdot)$.
  - Generate the translation table TT_i as follows. TT_i has the wire label $\boldsymbol{k}_{\mathsf{ow},\mathbf{r}_{\mathsf{ow}}}$ mapped to $C_i(x)$ and a random $(\mathsf{v}\lambda + 2)$-length string mapped to the complement of $C_i(x)$, where ow is the output wire of C_i.

Output $\Big(\{\mathsf{GC}_i, \mathbf{K}_i\}_{i \in [Q]} \Big)$.

Proof of Security. Consider the following hybrids.

$\underline{\mathbf{Hyb}_1}$: This corresponds to the real experiment. That is, the challenger computes the following: let $C_1, \ldots, C_Q$ be the circuit queries submitted by the adversary.

1. For $i \in [Q]$, Δ_i is a v-sized set sampled uniformly at random from $[T]$.

2. For every wire $w \in \mathcal{W}$, index $j \in [T]$, bit $b \in \{0,1\}$, sample $\mathbf{sd}^j_{w,b} \xleftarrow{\$} \{0,1\}^\lambda$ and sample $\mathbf{r}^j_w \xleftarrow{\$} \{0,1\}$.
3. For every wire $w \in \mathcal{W}$, index $j \in [T]$, bit $b \in \{0,1\}$, let $\boldsymbol{k}^j_{w,b} = \mathsf{PRG}\left(\mathbf{sd}^j_{w,b}\right)$.
4. For every $i \in [Q]$, generate the following:

$$(\mathsf{GC}_i, \mathbf{K}_i) \leftarrow \mathsf{CorrGarb}\left(\Delta_i, C_i, x, \{\mathbf{sd}^j_{w,b}, \boldsymbol{k}^j_{w,b}\}_{\substack{j\in[T], w\in\mathcal{W},\\ b\in\{0,1\}}}, \{\mathbf{r}^j_w\}_{j\in[T], w\in\mathcal{W}}\right)$$

Output $\left(\{\mathsf{GC}_i, \mathbf{K}_i\}_{i\in[Q]}\right)$.

Hyb$_2$: This is identical to the previous hybrid except for the following change:

1. For $i \in [Q]$, Δ_i is a v-sized set sampled uniformly at random from $[T]$. Check if for every $i^* \in [Q]$, $\Delta_{i^*} \setminus \left(\cup_{\substack{i\in[Q]\\ i\neq i^*}} \Delta_i\right) \neq \emptyset$. If this condition does not hold, output $\perp$.

Claim 5. *The statistical distance between* **Hyb**$_1$ *and* **Hyb**$_2$ *is negligible in* λ.

Proof. This proof of this claim follows from the following lemma shown by [24].

Lemma 3 (Cover-Freeness Lemma [24])**.** *Let $Q \in \mathbb{Z}_{>0}$. Set $T = \Theta(\mathsf{v}Q^2)$ and $\mathsf{v} = \Theta(\lambda)$. Sample Δ_i is a v-sized set sampled uniformly at random from $[T]$. Then, for every $i^* \in [T]$, the following holds with overwhelming probability: $\Delta_{i^*} \setminus \left(\bigcup_{\substack{i\in[Q],\\ i\neq i^*}} \Delta_i\right) \neq \emptyset$.*

□

Hyb$_{3.\mathbf{j}}$ for $\mathbf{j} \in [Q]$: We consider a hybrid simulator that simulates the first $\mathbf{j}-1$ garbled circuits and generates the rest of the garbled circuits honestly.

- For every $i \in [Q]$, sample a v-sized set $\Delta_i \subseteq [T]$ uniformly at random. Check if for every $i^* \in [Q]$, $\Delta_{i^*} \setminus \left(\cup_{\substack{i\in[Q]\\ i\neq i^*}} \Delta_i\right) \neq \emptyset$. If this condition does not hold, output $\perp$.
- For every wire $w \in \mathcal{W}$, sample the PRG seeds as follows: for every $j \in [T]$, sample $\mathbf{sd}^j_{w,0} \xleftarrow{\$} \{0,1\}^\lambda$.
- For every wire $w \in \mathcal{W}$, index $j \in [T]$, $\mathbf{r}^j_w \xleftarrow{\$} \{0,1\}$.
- For every wire $w \in \mathcal{W}$, index $j \in [T]$, bit $b \in \{0,1\}$, let $\boldsymbol{k}^j_{w,b} = \mathsf{PRG}\left(\mathbf{sd}^j_{w,b}\right)$.
- $\mathcal{A}$ submits the challenge input x. It then adaptively submits the circuit queries $(C_1, \ldots, C_Q)$. The hybrid simulator generates the i^{th} garbled circuit depends on the following two cases.

- If $i \geq \mathbf{j}$, the hybrid simulator computes the following:

$$(\mathsf{GC}_i, \mathbf{K}_i) \leftarrow \mathsf{CorrGarb}\left(\Delta_i, C_i, x \left\{\mathbf{sd}^j_{w,b}, \boldsymbol{k}^j_{w,b}\right\}_{\substack{j \in [\mathsf{v}], w \in \mathcal{W} \\ b \in \{0,1\}}}, \left\{\mathbf{r}^j_w\right\}_{j \in [T], w \in \mathcal{W}}\right)$$

- For every $i < \mathbf{j}$, generate the i^{th} simulated garbled circuit GC_i as follows: let $\Delta_i = \{j_1, \ldots, j_\mathsf{v}\}$.
  - For every wire $w \in \mathcal{W}$ and bit b, set $\mathbf{sd}_{w,b} = \mathbf{sd}^{j_1}_{w,b} \circ \cdots \circ \mathbf{sd}^{j_\mathsf{v}}_{w,b}$.
  - For every wire $w \in \mathcal{W}$, sample $\mathbf{r}_w \xleftarrow{\$} \{0,1\}$ uniformly at random.
  - Next, generate the input wire labels of GC_i. Let,

$$\mathbf{K}_i = \left((\mathbf{sd}_{\mathsf{iw}_1, \mathbf{r}_{\mathsf{iw}_1}}; \mathbf{r}_{\mathsf{iw}_1}) \circ \cdots \circ (\mathbf{sd}_{\mathsf{iw}_L, \mathbf{r}_{\mathsf{iw}_L}}; \mathbf{r}_{\mathsf{iw}_L})\right)$$

  - Generation of garbled tables: for every gate G in $\mathcal{U}(\cdot,\cdot)$, where $\mathcal{U}(\cdot,\cdot)$ is the universal circuit that takes as input (C, x) and outputs $C(x)$, generate a garbled table for G as follows. Let w_1, w_2 be the input wires and w_3, w_4 be the output wires of C. Set the $(\mathbf{r}_{w_1}, \mathbf{r}_{w_2})^{th}$ entry in the garbled table to be the following:

$$\left(\boldsymbol{k}^{\mathbf{r}_{w_2}}_{w_1, \mathbf{r}_{w_1}}\right) \oplus \left(\boldsymbol{k}^{\mathbf{r}_{w_1}}_{w_2, \mathbf{r}_{w_2}}\right) \oplus \left(\mathbf{sd}_{w_3, \mathbf{r}_{w_3}} \circ \mathbf{r}_{w_3} \circ \mathbf{sd}_{w_4, \mathbf{r}_{w_4}} \circ \mathbf{r}_{w_4}\right),$$

    where $\bigoplus_{j \in \Delta_i} \mathsf{PRG}(\mathbf{sd}^j_{w_1, \mathbf{r}_{w_1}}) = (\boldsymbol{k}^0_{w_1, \mathbf{r}_{w_1}}) \circ (\boldsymbol{k}^1_{w_1, \mathbf{r}_{w_1}})$ and $\bigoplus_{j \in \Delta_i} \mathsf{PRG}(\mathbf{sd}_{w_2, \mathbf{r}_{w_2}}) = (\boldsymbol{k}^0_{w_2, \mathbf{r}_{w_2}}) \circ (\boldsymbol{k}^1_{w_2, \mathbf{r}_{w_2}})$. The other three entries in the garbled table are set to be random strings of length $2\mathsf{v}\lambda + 2$.

    Construct GC_i to consist of garbled gates for every gate in $\mathcal{U}(\cdot,\cdot)$.
  - Generate the translation table TT_i as follows. TT_i has the wire label $\boldsymbol{k}_{\mathsf{ow}, \mathbf{r}_{\mathsf{ow}}}$ mapped to $C_i(x)$ and a random $(\mathsf{v}\lambda + 2)$-length string mapped to the complement of the $C_i(x)$.

Output $\left(\{\mathsf{GC}_i, \mathbf{K}_i\}_{i \in [Q]}\right)$.

Claim 6. *The output distributions of* $\mathbf{Hyb}_2$ *and* $\mathbf{Hyb}_{3.1}$ *are identically distributed.*

Claim 7. *The output distributions of* $\mathbf{Hyb}_{3.\mathbf{j}}$ *and* $\mathbf{Hyb}_{3.\mathbf{j}+1}$ *are computationally indistinguishable.*

We postpone the proof of the above claim to later.

$\mathbf{Hyb}_4$: This corresponds to the simulated experiment.

Claim 8. *The output distributions of* $\mathbf{Hyb}_{3.Q}$ *and* $\mathbf{Hyb}_4$ *are computationally indistinguishable.*

The proof of the above claim follows along the same lines as the proof of Claim 7.

Proof of Claim 7. We consider intermediate sub-hybrids to prove this claim. We assume that the universal circuit $\mathcal{U}(\cdot,\cdot)$ is layered, meaning that the circuit is divided into layers and the output wires of each layer (except the layer consisting of output gates) are input into the subsequent layer; note that this is without loss of generality. Let L be the number of layers in $\mathcal{U}(\cdot,\cdot)$.

$\underline{\mathsf{SubHyb}_{\mathbf{j}.1.\ell}\text{ for }\ell\in[L-1]}$: The garbled circuits for all indices $i\in[Q]$ and $i\neq\mathbf{j}$ are generated as in $\mathbf{Hyb}_{3.\mathbf{j}-1}$ (or $\mathbf{Hyb}_2$, if $\mathbf{j}=1$). We focus on the generation of $\mathsf{GC}_{\mathbf{j}}$. For every gate G in $\mathcal{U}(\cdot,\cdot)$, generate a garbled table for G as follows: suppose w_1,w_2 be the input wires of G and w_3,w_4 be the output wires of G. Let b_1^* be the value assigned to w_1, b_2^* assigned to w_2 during the computation of $C_{\mathbf{j}}(x)$.

- G is in a layer below the ℓ^{th} layer: In this case, set the $(b_1^*\oplus\mathbf{r}_{w_1},b_2^*\oplus\mathbf{r}_{w_2})^{th}$ entry in the garbled table of G to be the following:

$$\boldsymbol{k}^{\mathbf{r}_{w_2}}_{w_1,b_1^*\oplus\mathbf{r}_{w_1}}\oplus\boldsymbol{k}^{\mathbf{r}_{w_1}}_{w_2,b_2^*\oplus\mathbf{r}_{w_2}}\quad\oplus\quad\begin{array}{l}(\mathbf{sd}_{w_3,G(b_1^*,b_2^*)\oplus\mathbf{r}_{w_3}}\circ G(b_1^*,b_2^*)\oplus\mathbf{r}_{w_3}\\ \circ\mathbf{sd}_{w_4,G(b_1^*,b_2^*)\oplus\mathbf{r}_{w_4}}\circ G(b_1^*,b_2^*)\oplus\mathbf{r}_{w_4})\end{array}$$

  The rest of the entries in the garbled table are generated to be uniformly generated strings of appropriate length.

- G is in the ℓ^{th} layer or a layer above the ℓ^{th} layer: the garbled table for G is honestly generated.

The output distributions of $\mathbf{Hyb}_{3.\mathbf{j}}$ and $\mathsf{SubHyb}_{\mathbf{j}.1.1}$ are identically distributed.

$\underline{\mathsf{SubHyb}_{\mathbf{j}.2.\ell}\text{ for }\ell\in[L-1]}$: In the previous hybrid, for every wire w input into the ℓ layer, let $\mathbf{sd}_{w,b\oplus\mathbf{r}_w}\circ b\oplus\mathbf{r}_w$ be the key associated with w (where b is the value carried by w during the execution of $C_{\mathbf{j}}(x)$) used in the generation of garbled tables for the layers below the ℓ^{th} layer; in particular, $\mathbf{sd}_{w,\overline{b\oplus\mathbf{r}_w}}$ is not encoded in any of the garbled tables below the ℓ^{th} layer.

Let $j^*\in[T]$ be an index such that $j^*\in\Delta_{\mathbf{j}}\setminus\left(\bigcup_{i\in[T],\,i\neq\mathbf{j}}\Delta_i\right)$. Generate $(\boldsymbol{k}_{w,\overline{b\oplus\mathbf{r}_w}})^{j^*}\xleftarrow{\$}\{0,1\}^{2\mathsf{v}\lambda+2}$. This is the only change in the generation of $\mathsf{GC}_{\mathbf{j}}$. The rest of the steps in the generation of $\mathsf{GC}_{\mathbf{j}}$ is identical to the previous hybrid.

The output distributions of the hybrids $\mathsf{SubHyb}_{\mathbf{j}.1.\ell}$ and $\mathsf{SubHyb}_{\mathbf{j}.2.\ell}$ are computationally indistinguishable from the security of pseudorandom generators; in particular, we need to invoke the security of pseudorandom generators as many times as the number of wires input to the ℓ^{th} layer.

Remark 6. *Note that in the above hybrid, if we picked an index $j^*\in\Delta_{\mathbf{j}}\cap\Delta_i$, for some $i\neq\mathbf{j}$ then the output distributions of the hybrids would no longer be computationally indistinguishable. The reason is that it is possible that $\mathbf{sd}^{j^*}_{w,\overline{b\oplus\mathbf{r}_w}}$ (where $\mathbf{r}_w$ is defined as part of generation of $\mathsf{GC}_{\mathbf{j}}$) is used in the generation of GC_i and thus we no longer can invoke the security of PRGs. Thus, the cover-freeness of the sets $\Delta_1,\ldots,\Delta_T$ is crucially used above.*

Also, the output distributions of $\mathsf{SubHyb}_{\mathbf{j}.2.\ell}$ and $\mathsf{SubHyb}_{\mathbf{j}.1.\ell+1}$ are identically distributed.

$\underline{\mathsf{SubHyb}_{\mathbf{j}.3}}$: The garbled circuits for all indices $i\in[Q]$ and $i\neq\mathbf{j}$ are generated as in $\mathbf{Hyb}_{3.\mathbf{j}-1}$ (or $\mathbf{Hyb}_2$, if $\mathbf{j}=1$). We focus on the generation of $\mathsf{GC}_{\mathbf{j}}$. For every gate G in $\mathcal{U}(\cdot,\cdot)$, generate a garbled table for G as follows: suppose w_1,w_2 be the input wires of G and w_3,w_4 be the output wires of G.

- Set the $(\mathbf{r}_{w_1},\mathbf{r}_{w_2})^{th}$ entry in the garbled table of G, where $\mathbf{r}_w$ is a random mask associated with wire $w \in \mathcal{W}$, to be the following:

$$\boldsymbol{k}_{w_1,\mathbf{r}_{w_1}}^{\mathbf{r}_{w_2}} \oplus \boldsymbol{k}_{w_2,\mathbf{r}_{w_2}}^{\mathbf{r}_{w_1}} \oplus \left(\mathbf{sd}_{w_3,\mathbf{r}_{w_3}} \circ \mathbf{r}_{w_3} \circ \mathbf{sd}_{w_4,\mathbf{r}_{w_4}} \circ \mathbf{r}_{w_4}\right)$$

  The rest of the entries in the garbled table are generated to be uniformly generated strings of appropriate length.

The difference between $\mathsf{SubHyb}_{\mathbf{j}.2.L}$ and $\mathsf{SubHyb}_{\mathbf{j}.3}$ is: in $\mathsf{SubHyb}_{\mathbf{j}.2.L}$, the garbled table of G is generated by encoding the label of the output wire of G in the $(b_1^* \oplus \mathbf{r}_{w_1}, b_2^* \oplus \mathbf{r}_{w_2})^{th}$ entry, while in $\mathsf{SubHyb}_{\mathbf{j}.3}$, the label of the output wire is encoded in the $(\mathbf{r}_{w_1},\mathbf{r}_{w_2})^{th}$ entry. Since $\mathbf{r}_{w_1}$ is picked at random and used only to mask b_1^*, it follows that the generation processes of the garbled tables in the two hybrids are identical. This shows that the output distributions of $\mathsf{SubHyb}_{\mathbf{j}.2.L}$ and $\mathsf{SubHyb}_{\mathbf{j}.3}$ are identically distributed.

Finally, observe that the output distributions of $\mathsf{SubHyb}_{\mathbf{j}.3}$ and $\mathbf{Hyb}_{3.\mathbf{j}+1}$ are identically distributed.

This proves that the output distributions of $\mathbf{Hyb}_{3.\mathbf{j}}$ and $\mathbf{Hyb}_{3.\mathbf{j}+1}$ are computationally indistinguishable.

□

5.2.2 Construction

We are now ready to present the construction of a protocol in the client-server framework.

Lemma 4. *Let $Q \in \mathbb{Z}_{>0}$. Let N, n, t be any positive integers such that $N > \Theta(\lambda Q^2)$ and $t < \lfloor \frac{n}{3} \rfloor$ and $n < N$. Assuming the existence of pseudorandom generators, there exists a protocol Π in the client-server framework for* P/Poly*, parameterized by N, n and t, for* P/Poly*.*

Moreover, the construction of this protocol makes black box usage of the pseudorandom generator.

Proof. The main ingredient in the construction is correlated garbling; we denote the correlated garbling algorithms by $(\mathsf{CorrGarb}, \mathsf{CorrEval})$. Another tool we use in this construction is a two-round n-party computation protocol securely computing cubic polynomials and tolerating $\lfloor \frac{n}{3} \rfloor$ corruptions. The celebrated work of Ben-Or, Goldwasser and Widgerson [9] can be adapted to yield such a protocol. At a high level, the protocol works as follows: every party shares its input using a Shamir secret sharing scheme. At the end of first round, every party can homomorphically compute a cubic polynomial to obtain a sharing of the output. In the second round, all the parties exchange their respective shares and at the end of the second round, they can reconstruct the secret. Moreover, for the construction of client-server protocol, we require that the first round of this protocol can be re-used for multiple executions. Inspired from the ideas in [24], we show how to achieve reusability of the first round of this protocol. Informally, we achieve reusability by exchanging sufficiently many shares of zeroes in the first round and then in the second round, every party combines a random subset of shares of zeroes and adds this result to its second round message.

Using this protocol, we compute the client-server protocol as follows:

- The client generates the first round messages of all the parties. All the first round messages form the input encodings of the client-server protocol.

- The encoding of a circuit C is just (C, Δ), where Δ determines the subset of random shares of zeroes to be combined.
- The local computation corresponds to the computation of the second round message of the reusable two-round secure protocol we constructed above.
- Finally, the decoding procedure takes as input all the second round messages and computes the reconstruction of the secure MPC protocol.

We now describe the construction. Consider a pseudorandom generator $\mathsf{PRG} : \{0,1\}^{\lambda} \to \{0,1\}^{\mathsf{v}\lambda+2}$. Set $T = \Theta(\mathsf{v}Q^2)$ and $\mathsf{v} = \Theta(\lambda)$. Consider a binary extension field $\mathbb{F}$ of size at least n.

- $\mathsf{InpEnc}\left(1^{\lambda}, 1^{Q}, 1^{s}, x\right)$: On input the security parameter λ, query bound Q, circuit size s, input x of length ℓ, do the following:
  - Consider an universal circuit $\mathcal{U}$ that takes as input a circuit C of size s, input x and outputs $C(x)$. Let the size of $\mathcal{U}$ be s'. The input wires of $\mathcal{U}$ are partitioned into sets $\mathcal{I}_1$ and $\mathcal{I}_2$ such that the wires in $\mathcal{I}_1$ are assigned with C and the wires in $\mathcal{I}_2$ are assigned with x. The set of all the wires in $\mathcal{U}$ will be denoted by $\mathcal{W}$.
  - For every wire $w \in \mathcal{W}$, $b \in \{0,1\}$, $j \in [T]$, generate $\mathbf{sd}^{j}_{w,b} \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$ and let $\boldsymbol{k}^{j}_{w,b} = \mathsf{PRG}(\mathbf{sd}^{j}_{w,b})$.
  - For every wire $w \in \mathcal{W}$, $j \in [T]$, sample $\mathbf{r}^{j}_{w} \overset{\$}{\leftarrow} \{0,1\}$.
  - For every wire w in $\mathcal{I}_1$, $b \in \{0,1\}$, sample a degree-t polynomial $\delta_{w,b}(\cdot)$ over $\mathbb{F}$ whose constant term in $\delta_{w,b}(\cdot)$ is b.
    Let $\boldsymbol{E}^{\delta}_{\boldsymbol{u}} = \{\delta_{w,b}(\boldsymbol{u})\}_{w \in \mathcal{I}_1, b \in \{0,1\}}$.
  - For every wire w in $\mathcal{I}_2$, $h \in [n]$, sample a degree-t polynomial $\mu_{w,h}(\cdot)$ over $\mathbb{F}$ whose constant term is x_h, where the hth bit of x is assigned to wire w.
    Let $\boldsymbol{E}^{\mu}_{\boldsymbol{u}} = \{\mu_{w,h}(\boldsymbol{u})\}_{\substack{w \in \mathcal{I}_2, \\ h \in [\ell]}}$.
  - For every wire $w \in \mathcal{W}$, $b \in \{0,1\}$, $j \in [T]$, $h \in [\lambda]$, sample a random degree-t polynomial $\zeta^{j}_{w,b,h}(\cdot)$ over $\mathbb{F}$ whose constant term in $\zeta^{j}_{w,b,h}$ is the hth bit of $\mathbf{sd}^{j}_{w,b}$.
    Let $\boldsymbol{E}^{\zeta}_{\boldsymbol{u}} = \left\{\zeta^{j}_{w,b,h}(\boldsymbol{u})\right\}_{\substack{w \in \mathcal{W},\ b \in \{0,1\} \\ h \in [\lambda], j \in [T]}}$.
  - For every wire $w \in \mathcal{W}$, $b \in \{0,1\}$, $j \in [T]$, $h \in [\mathsf{v}\lambda+2]$, sample a random degree-t polynomial $\xi^{j}_{w,b,h}(\cdot)$ over $\mathbb{F}$ whose constant term in $\xi^{j}_{w,b,h}(\cdot)$ is the hth bit of $\boldsymbol{k}^{j}_{w,b}$.
    Let $\boldsymbol{E}^{\xi}_{\boldsymbol{u}} = \left\{\xi^{j}_{w,b,h}\left(\boldsymbol{u}\right)\right\}_{\substack{w \in \mathcal{W}, b \in \{0,1\} \\ h \in [\mathsf{v}\lambda+2], j \in [T]}}$.
  - For every wire $w \in \mathcal{W}$, $j \in [T]$, sample a degree-t polynomial η^{j}_{w} over $\mathbb{F}$ whose constant term in $\eta^{j}_{w}(\cdot)$ is $\mathbf{r}^{j}_{w}$.
    Let $\boldsymbol{E}^{\eta}_{\boldsymbol{u}} = \left\{\eta^{j}_{w}(\boldsymbol{u})\right\}_{w \in \mathcal{W}, j \in [T]}$.

- For every $j \in [T]$, $h \in [s'']$, sample a degree-n polynomial $\vartheta_h^j(\cdot)$ over $\mathbb{F}$ whose constant term in $\vartheta_h^j(\cdot)$ is 0. We define s'' later.
  Let $\boldsymbol{E}_{\boldsymbol{u}}^{\vartheta} = \left\{\vartheta_h^j(\boldsymbol{u})\right\}_{h\in[s'']j\in[T]}$.
- For every $\boldsymbol{u} \in [N]$, compute the $\boldsymbol{u}^{th}$ input encoding $\widehat{x}^{\boldsymbol{u}}$ as follows:

$$\widehat{x}^{\boldsymbol{u}} = \left(\boldsymbol{E}_{\boldsymbol{u}}^{\delta}, \boldsymbol{E}_{\boldsymbol{u}}^{\mu}, \boldsymbol{E}_{\boldsymbol{u}}^{\zeta}, \boldsymbol{E}_{\boldsymbol{u}}^{\xi}, \boldsymbol{E}_{\boldsymbol{u}}^{\eta}, \boldsymbol{E}_{\boldsymbol{u}}^{\vartheta}\right)$$

Output $\left(\widehat{x}^1, \ldots, \widehat{x}^N\right)$.

- $\mathsf{CktEnc}\left(1^\lambda, 1^Q, 1^s, C\right)$: On input the security parameter λ, query bound Q, size bound s, circuit C, sample a v-sized set $\Delta \subseteq [T]$, set the circuit encoding $\widehat{C}^{\boldsymbol{u}} = (C, \Delta)$, for every $\boldsymbol{u} \in [N]$. That is, all the circuit encodings are set to be the same value. Output $(\widehat{C}^1, \ldots, \widehat{C}^N)$.
- $\mathsf{Local}\left(\widehat{C}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}}\right)$: On input the circuit encoding $\widehat{C}^{\boldsymbol{u}}$, input encoding $\widehat{x}^{\boldsymbol{u}}$, do the following:
  - Parse $\widehat{x}^{\boldsymbol{u}}$ as $\left(\boldsymbol{E}_{\boldsymbol{u}}^{\delta}, \boldsymbol{E}_{\boldsymbol{u}}^{\mu}, \boldsymbol{E}_{\boldsymbol{u}}^{\zeta}, \boldsymbol{E}_{\boldsymbol{u}}^{\xi}, \boldsymbol{E}_{\boldsymbol{u}}^{\eta}, \boldsymbol{E}_{\boldsymbol{u}}^{\vartheta}\right)$.
  - Let $\boldsymbol{b}_1 \cdots \boldsymbol{b}_s$ be the binary representation of C. Let $\boldsymbol{E}_{\boldsymbol{u},C}^{\delta} = \{\delta_{w,\boldsymbol{b}_h}(\boldsymbol{u})\}_{w\in\mathcal{I}_1, h\in[s]}$.
  - For every $h \in [s'']$, let $\mathsf{zrsh}_h^{\boldsymbol{u}} = \sum_{j\in\Delta} \vartheta_h^j(\boldsymbol{u})$.
  - Compute the polynomial $\mathsf{CorrGarb}$ on input $\widehat{x}^{\boldsymbol{u}}$ and circuit encoding (C, Δ) and add $\mathsf{zrsh}^{\boldsymbol{u}}$ to obtain $\widehat{y}^{\boldsymbol{u}}$; that is, for every $h \in [s'']$,

$$\widehat{y}_h^{\boldsymbol{u}} = \left(\mathsf{CorrGarb}\left(\Delta, \boldsymbol{E}_{\boldsymbol{u},C}^{\delta}, \boldsymbol{E}_{\boldsymbol{u}}^{\mu}, \boldsymbol{E}_{\boldsymbol{u}}^{\zeta}, \boldsymbol{E}_{\boldsymbol{u}}^{\xi}, \boldsymbol{E}_{\boldsymbol{u}}^{\eta}\right)\right)_h + \mathsf{zrsh}_h^{\boldsymbol{u}}$$

  Output the encodings, $\widehat{y}^{\boldsymbol{u}} = (\widehat{y}_1^{\boldsymbol{u}}, \ldots, \widehat{y}_{s''}^{\boldsymbol{u}}) \in \mathbb{F}^{s''}$.
- $\mathsf{Decode}\left(\{\widehat{y}^{\boldsymbol{u}}\}_{\boldsymbol{u}\in\mathbf{S}}, \mathbf{S}\right)$: On input evaluated values $\{\widehat{y}^{\boldsymbol{u}}\}_{\boldsymbol{u}\in S}$, set $\mathbf{S}$, reconstruct the polynomials $\{p_h\}_{h\in[s'']}$ such that $p_h(\boldsymbol{u}) = (\widehat{y}^{\boldsymbol{u}})_h$, where $(\widehat{y}^{\boldsymbol{u}})_h$ is the h^{th} element in the vector $\widehat{y}^{\boldsymbol{u}}$. Let $(\mathsf{GC}, \mathbf{K}) = p_1(0) \circ \cdots \circ p_{s''}(0)$. That is, $s'' = |\mathsf{GC}| + |\mathbf{K}|$. Compute $\mathsf{CorrEval}(\mathsf{GC}, \mathbf{K})$ to obtain the result y.

Correctness. Consider a circuit C of size s and an input x. Consider a set $\mathbf{S} \subseteq [N]$ and $|\mathbf{S}| = n$. Let $\boldsymbol{b}_1 \cdots \boldsymbol{b}_s$ be the binary representation of C. Generate the input encodings and the circuit encoding as follows:

- For every $\boldsymbol{u} \in [N]$, $\widehat{x}^{\boldsymbol{u}} \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, 1^s, x)$. Parse $\widehat{x}^{\boldsymbol{u}}$ as,

$$\widehat{x}^{\boldsymbol{u}} = \left(\boldsymbol{E}_{\boldsymbol{u}}^{\delta}, \boldsymbol{E}_{\boldsymbol{u}}^{\mu}, \boldsymbol{E}_{\boldsymbol{u}}^{\zeta}, \boldsymbol{E}_{\boldsymbol{u}}^{\xi}, \boldsymbol{E}_{\boldsymbol{u}}^{\eta}, \boldsymbol{E}_{\boldsymbol{u}}^{\vartheta}\right),$$

  where:
  - $\boldsymbol{E}_{0,C}^{\delta} = C$ ($\boldsymbol{E}_{\boldsymbol{u},C}^{\delta}$ is defined as above).
  - $\boldsymbol{E}_0^{\mu} = x$.

  - $\boldsymbol{E}_0^{\zeta} = \left\{ \mathbf{sd}_{w,b}^j \right\}_{\substack{w \in \mathcal{W}, b \in \{0,1\} \\ j \in [T]}}$.
  - $\boldsymbol{E}_0^{\xi} = \left\{ \boldsymbol{k}_{w,b}^j \right\}_{\substack{w \in \mathcal{W}, b \in \{0,1\} \\ j \in [T]}}$.
  - $\boldsymbol{E}_0^{\eta} = \left\{ \mathbf{r}_w^j \right\}_{\substack{w \in \mathcal{W} \\ j \in [T]}}$.
  - $\boldsymbol{E}_0^{\vartheta} = 0$.

- $\left(\widehat{C}^1, \ldots, \widehat{C}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C)$.
- For every $\boldsymbol{u} \in [N]$, $\widehat{y}^{\boldsymbol{u}} = \mathsf{Local}\left(\widehat{C}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}}\right)$

We examine the computation of Decode on $\left(\{\widehat{y}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathbf{S}}, \mathbf{S}\right)$. Let $\mathsf{zrsh}^0 = \mathsf{zrsh}_1^0 \circ \cdots \circ \mathsf{zrsh}_h^0$

$$\begin{aligned}
\mathsf{Decode}\left(\{\widehat{y}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathbf{S}}, \mathbf{S}\right) &= \mathsf{CorrEval}\left(\mathsf{GC}, \mathbf{K}\right) \\
&= \mathsf{Eval}\left(p_1(0) \circ \cdots \circ p_{s'}(0)\right) \\
&= \mathsf{CorrEval}\left(\mathsf{CorrGarb}\left(\Delta, \boldsymbol{E}_{0,C}^{\delta}, \boldsymbol{E}_0^{\mu}, \boldsymbol{E}_0^{\zeta}, \boldsymbol{E}_0^{\xi}, \boldsymbol{E}_0^{\eta}\right) + \mathsf{zrsh}^0\right) \\
&= \mathsf{CorrEval}\left(\mathsf{CorrGarb}\left(\Delta, C, x, \left\{\mathbf{sd}_{w,b}^j, \boldsymbol{k}_{w,b}^j\right\}_{\substack{w \in \mathcal{W}, b \in \{0,1\} \\ j \in [T]}}, \left\{\mathbf{r}_w^j\right\}_{\substack{w \in \mathcal{W} \\ j \in [T]}}\right)\right) \\
&= C(x)
\end{aligned}$$

The last equation follows from the correctness of $(\mathsf{CorrGarb}, \mathsf{Eval})$.

Security. We describe the simulator $\mathsf{Sim}_{\mathsf{csf}}$. Let $\mathcal{A}$ be the adversary interacting with $\mathsf{Sim}_{\mathsf{csf}}$ in the simulated experiment. Let $\mathsf{Sim}_{\mathsf{Corr}}$ be the simulator associated with $(\mathsf{CorrGarb}, \mathsf{CorrEval})$ as guaranteed by the correlated garbling lemma (Lemma 2).

$\mathsf{Sim}_{\mathsf{csf}}$ proceeds as follows:

- It receives $\left(Q, s, N, n, t, \mathcal{S}_{\mathsf{corr}}, \{\mathbf{S}_i\}_{i \in [Q]}\right)$ from $\mathcal{A}$. If $|\mathcal{S}_{\mathsf{corr}}| > t$ then abort.
- $\mathcal{A}$ submits Q_1 circuit queries $C_1, \ldots, C_{Q_1}$ adaptively. For every $i \in [Q_1]$, sample the v-sized set Δ_i uniformly at random from $[T]$. If there exists $i^* \in [Q_1]$ such that $\Delta_{i^*} \setminus \left(\bigcup_{\substack{i \neq i^* \\ i \in [Q_1]}} \Delta_i\right) = \emptyset$ then output $\perp$. Otherwise, set $\widehat{C_i}^{\boldsymbol{u}} = (C_i, \Delta_i)$, for every $\boldsymbol{u} \in \mathbf{S}_i$. Send $\left(\left\{\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right)\right\}_{i \in [Q_1]}\right)$ to $\mathcal{A}$.
- Let $\mathcal{V} = \{(C_i, C_i(x)) : i \in [Q_1]\}$, where x is the challenge message output by $\mathcal{A}$. On input $(1^{|x|}, \mathcal{S}_{\mathsf{corr}}, \mathcal{V})$, generate the following:
  - For every $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}$, generate the $\boldsymbol{u}^{th}$ simulated input encoding as $\widehat{x}^{\boldsymbol{u}} \xleftarrow{\$} \mathbb{F}^L$, where the honestly generated $\boldsymbol{u}^{th}$ input encoding of x is a vector in $\mathbb{F}$ of length L.
  - For every $i \in [Q_1]$, compute $(\mathsf{GC}_i, \mathbf{K}_i) \leftarrow \mathsf{Sim}_{\mathsf{Corr}}\left(C_i, C_i(x)\right)$.

  - For every $i \in [Q_1]$, for every $h \in [s'']$, sample random degree-n polynomials $p_h^i(\cdot)$ subject to the condition that $\left(p_1^i(0) \circ \cdots \circ p_{s''}^i(0)\right) = (\mathsf{GC}_i, \mathbf{K}_i)$ for every $i \in [Q_1]$. Set $\widehat{y}_i^{\boldsymbol{u}} = (p_1^i(\boldsymbol{u}), \ldots, p_{s''}^i(\boldsymbol{u}))$ for every $\boldsymbol{u} \in \mathbf{S}_i$.

  Send $\left(\{\widehat{x}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}}, \{\widehat{y}_i^{\boldsymbol{u}}\}_{i \in [Q_1], \boldsymbol{u} \in \mathbf{S}_i}\right)$ to adversary.

- $\mathcal{A}$ submits $Q - Q_1$ circuit queries $C_{Q_1+1}, \ldots, C_Q$ adaptively. For $i = Q_1 + 1, \ldots, Q$, on input $(i, C_i, C_i(x))$, do the following:
  - Sample a v-sized set Δ_i uniformly at random from $[T]$. If there exists $i^* \in \{Q_1+1, \ldots, Q\}$ such that $\Delta_{i^*} \backslash \left(\bigcup_{\substack{i' \neq i^* \\ i' \in [Q]}} \Delta'_i\right) = \emptyset$ then output $\perp$. Otherwise, set $\widehat{C_i}^{\boldsymbol{u}} = (C_i, \Delta_i)$, for every $\boldsymbol{u} \in \mathbf{S}_i$. Send $\left(\left\{\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right)\right\}_{i \in \{Q_1+1, \ldots, Q\}}\right)$ to $\mathcal{A}$.
  - Compute $(\mathsf{GC}_i, \mathbf{K}_i) \leftarrow \mathsf{Sim}_{\mathsf{Corr}}(C_i, C_i(x))$.
  - For every $h \in [s'']$, sample random degree-n polynomials $p_h^i(\cdot)$ subject to the condition that $\left(p_1^i(0) \circ \cdots \circ p_{s''}^i(0)\right) = (\mathsf{GC}_i, \mathbf{K}_i)$. Set $\widehat{y}_i^{\boldsymbol{u}} = (p_1^i(\boldsymbol{u}), \ldots, p_{s''}^i(\boldsymbol{u}))$ for every $\boldsymbol{u} \in \mathbf{S}_i$.
  - Send $(\{\widehat{y}_i^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathbf{S}_i})$ to $\mathcal{A}$.

We now describe the hybrids.

$\mathbf{Hyb}_1$: This corresponds to the real experiment.

$\mathbf{Hyb}_2$: This hybrid is identical to the previous hybrid except for the following change: let $(C_1, \ldots, C_Q)$ be the set of adaptive circuit queries made by the adversary and let $\widehat{C_i}^{\boldsymbol{u}} = (C_i, \Delta_i)$ be the circuit encoding issued for every $i \in [Q]$, $\boldsymbol{u} \in [\mathbf{S}_i]$. If there exists $i^* \in [Q]$ such that $\Delta_{i^*} \backslash \left(\bigcup_{\substack{i \neq i^* \\ i \in [Q]}} \Delta_i\right) = \emptyset$ then output $\perp$.

Claim 9. *The statistical difference between the output distributions of* $\mathbf{Hyb}_1$ *and* $\mathbf{Hyb}_2$ *is* $\varepsilon = \mathsf{negl}(\lambda)$*, for some negligible function* negl.

The proof of the above claim follows from lemma 3.

$\mathbf{Hyb}_3$: This hybrid is identical to the previous hybrid except that the distributed garbled circuits are generated differently: for the i^{th} circuit query C_i,

- First generate the garbled circuit using $\mathsf{CorrGarb}$ as follows:

$$(\mathsf{GC}_i, \mathbf{K}_i) \leftarrow \mathsf{CorrGarb}\left(\Delta_i, C_i, x, \left\{\mathbf{sd}_{w,b}^j, \boldsymbol{k}_{w,b}^j\right\}_{\substack{w \in \mathcal{W}, j \in [T] \\ b \in \{0,1\}}}, \left\{\mathbf{r}_w^j\right\}_{w \in \mathcal{W}, j \in [T]}\right)$$

- For every $h \in [s'']$, sample random degree-n polynomials $p_h^i(\cdot)$ subject to the condition that $\left(p_1^i(0) \circ \cdots \circ p_{s''}^i(0)\right) = (\mathsf{GC}_i, \mathbf{K}_i)$. Set the distributed garbled circuit to be $\widehat{y}_i^{\boldsymbol{u}} = (p_1^i(\boldsymbol{u}), \ldots, p_{s''}^i(\boldsymbol{u}))$ for every $\boldsymbol{u} \in \mathbf{S}_i$.

Claim 10. *The output distributions of* $\mathbf{Hyb}_2$ *and* $\mathbf{Hyb}_3$ *are identically distributed.*

Proof. For every $i^* \in [Q]$, there exists $j \in [T]$ such that $j^* \in \Delta_{i^*} \setminus \left(\bigcup_{\substack{i \in [T] \\ i \neq i^*}} \Delta_i \right)$ (otherwise, the experiment aborts). This means that the shares, generated as part of the input encoding, corresponding to the j^{th} index are not used in any circuit query other than the $(i^*)^{th}$ circuit query. Thus, the distribution of honestly generated encodings $\{\widehat{y}_{i^*}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathbf{S}_{i^*}}$ are identically distributed to $\left\{(p_1^{i^*}(\boldsymbol{u}), \ldots, p_{s''}^{i^*}(\boldsymbol{u}))\right\}_{\boldsymbol{u} \in \mathbf{S}_{i^*}}$, where $p_h^{i^*}(\cdot)$ is a random degree-n polynomial subject to the condition that $\left(p_1^{i^*}(0) \circ \cdots \circ p_{s''}^{i^*}(0)\right) = (\mathsf{GC}_{i^*}, \mathbf{K}_{i^*})$. □

Hyb$_4$: This corresponds to the simulated experiment.

The output distributions of **Hyb**$_3$ and **Hyb**$_4$ are computationally indistinguishable from the correlated garbling lemma 2. □

5.3 Bounded-Key FE for P/Poly from Client-Server Framework

We now present a construction of a bounded-key functional encryption for all polynomial-sized circuits from a protocol in the client-server framework.

Theorem 5. *There exists a public-key (resp., private-key) adaptively secure bounded-key functional encryption scheme* BFE *for* P/Poly *assuming,*

- *A public-key (resp., private-key) adaptively secure single-key functional encryption scheme* 1fe *for* P/Poly *and,*
- *A protocol for* P/Poly *in the client-server framework, denoted by* $\Pi = (\mathsf{InpEnc}, \mathsf{CktEnc}, \mathsf{Local}, \mathsf{Decode})$.

Proof. We focus on the public-key setting; the construction and the analysis for the private-key setting is identical. We describe the algorithms of BFE below. Let the protocol in the client-server framework be parameterized by $t = \Theta(Q\lambda)$, $N = \Theta(Q^2t^2)$ and $n = \Theta(t)$, where Q is the query bound defined as part of the scheme.

- $\underline{\mathsf{Setup}(1^\lambda, 1^Q, 1^s)}$: On input security parameter λ, query bound Q, circuit size s, generate $(\mathsf{msk}_i, \mathsf{mpk}_i) \leftarrow \mathsf{1fe.Setup}(1^\lambda, 1^s)$ for $i \in [N]$. Output the following:

$$\mathsf{MSK} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_N),\ \mathsf{MPK} = (Q,\ \mathsf{mpk}_1, \ldots, \mathsf{mpk}_N)$$

- $\underline{\mathsf{KeyGen}(\mathsf{MSK}, C)}$: On input master secret key MSK, circuit C,
  - Sample a set $\mathbf{S} \xleftarrow{\$} [N]$, of size n, uniformly at random.
  - Compute $\left(\widehat{C}^1, \ldots, \widehat{C}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C)$.
  - Let $\mathsf{E}^{\boldsymbol{u}}(\cdot) = \mathsf{Local}(\widehat{C}^{\boldsymbol{u}}, \cdot)$. Generate a functional key for $\mathsf{E}^{\boldsymbol{u}}$; that is, compute $\mathsf{sk}_{\mathsf{E}^{\boldsymbol{u}}} \leftarrow \mathsf{1fe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, \mathsf{E}^{\boldsymbol{u}})$ for every $\boldsymbol{u} \in \mathbf{S}$.

  Output $\mathsf{SK}_C = \left(\mathbf{S},\ \{\mathsf{sk}_{\mathsf{E}^{\boldsymbol{u}}}\}_{\boldsymbol{u} \in \mathbf{S}}\right)$.

- $\underline{\mathsf{Enc}(\mathsf{MPK}, x)}$: On input master public key MPK,

  - Compute $(\widehat{x}^1, \ldots, \widehat{x}^N) \leftarrow \mathsf{InpEnc}\left(1^\lambda, 1^Q, 1^s, x\right)$.
  - For every $i \in [N]$, compute $\mathsf{ct}_i \leftarrow \mathsf{FE.Enc}\left(\mathsf{mpk}_i, \widehat{x}^i\right)$.

  Output $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$.

- $\underline{\mathsf{Dec}(\mathsf{SK}_C, \mathsf{CT})}$: On input functional key $\mathsf{SK}_C = \left(\mathbf{S},\ \{\mathsf{sk}_{\mathsf{E}^{\boldsymbol{u}}}\}_{\boldsymbol{u}\in\mathbf{S}}\right)$, ciphertext $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$,
  - For every $\boldsymbol{u} \in \mathbf{S}$, compute $\widehat{y}^{\boldsymbol{u}} \leftarrow \mathsf{1fe.Dec}(\mathsf{sk}_{\mathsf{E}^{\boldsymbol{u}}}, \mathsf{ct}_{\boldsymbol{u}})$.
  - Compute $\mathsf{Decode}\left(\{\widehat{y}^{\boldsymbol{u}}\}_{\boldsymbol{u}\in\mathbf{S}}, \mathbf{S}\right)$ to obtain y.

  Output y.

Correctness. Consider a circuit C and an input x. Suppose $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{MPK}, x)$ and $\mathsf{SK}_C \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, C)$. Let $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$ and $\mathsf{SK}_C = \left(\mathbf{S},\ \{\mathsf{sk}_{\mathsf{E}^{\boldsymbol{u}}}\}_{\boldsymbol{u}\in\mathbf{S}}\right)$. By the correctness of $\mathsf{1fe}$, $\mathsf{1fe.Dec}(\mathsf{sk}_{\mathsf{E}^{\boldsymbol{u}}}, \mathsf{ct}^{\boldsymbol{u}}) = \mathsf{Local}(\widehat{C}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}})$ for every $\boldsymbol{u} \in \mathbf{S}$. From the correctness of Π, it follows that $\mathsf{Decode}\left(\left\{\mathsf{Local}(\widehat{C}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}})\right\}_{\boldsymbol{u}\in\mathbf{S}}, \mathbf{S}\right) = C(x)$.

Security. The proof proceeds in the following steps:

- First, we argue that with overwhelming probability, the union of pairwise intersections of the sets $\{\mathbf{S}_i\}_{i\in[Q]}$ has size at most t.
- This means that at most t instantiations of $\mathsf{1fe}$ are compromised; if there is an index $\boldsymbol{u} \in [N]$ such that $\boldsymbol{u}$ is in $\mathbf{S}_i \cap \mathbf{S}_j$ then this means that two functional keys corresponding to the i^{th} and j^{th} query are issued with respect to the $\boldsymbol{u}^{th}$ instantiation of $\mathsf{1fe}$. Furthermore, the input encodings of the client-server protocol encrypted in the compromised $\mathsf{1fe}$ instantiations are in turn compromised.
- Now, we simulate all the un-compromised instantiations of $\mathsf{1fe}$ using the simulator of $\mathsf{1fe}$. Moreover, since at most t input encodings of the client-server protocol are leaked, we can invoke the security of the client-server protocol to prove the security of our bounded-key FE scheme.

We describe the simulator SIM, interacting with adversary $\mathcal{A}$, below. Since we invoke $\mathsf{1fe}$ for N times, we consider N instantiations of the stateful simulator of $\mathsf{1fe}$, denoted by $\mathsf{sim}_1, \ldots, \mathsf{sim}_N$. Let $\mathsf{Sim}_{\mathsf{csf}}$ be the stateful simulator of Π.

SIM proceeds as follows:

1. It receives the query bound Q and the maximum circuit size s from $\mathcal{A}$.
2. For every $i \in [N]$, sample a set $\mathbf{S}$ of size n uniformly at random from $[N]$. Construct a set $\mathcal{S}_{\mathsf{corr}}$ as follows: include an index $\boldsymbol{u} \in [N]$ in the set $\mathcal{S}_{\mathsf{corr}}$ if and only if $\boldsymbol{u} \in \mathbf{S}_i \cap \mathbf{S}_j$, for some $i, j \in [Q]$ and $i \neq j$. If $|\mathcal{S}_{\mathsf{corr}}| > t$, output $\perp$.
3. Compute the master public key as follows: for every $\boldsymbol{u} \in [N]$,
   - If $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}$, compute $(\mathsf{msk}_{\boldsymbol{u}}, \mathsf{mpk}_{\boldsymbol{u}}) \leftarrow \mathsf{1fe.Setup}(1^\lambda, 1^Q, 1^s)$.

- If $\boldsymbol{u} \notin \mathcal{S}_{\mathsf{corr}}$, compute $\mathsf{mpk}_{\boldsymbol{u}} \leftarrow \mathsf{sim}_{\boldsymbol{u}}(1^\lambda, 1^s)$.

Set $\mathsf{MPK} = (\mathsf{mpk}_1, \ldots, \mathsf{mpk}_N)$. Send MPK to $\mathcal{A}$.

4. $\mathcal{A}$ makes a total of Q circuit queries throughout the experiment. Before the challenge input query, it makes Q_1 circuit queries. Initialize the simulator $\mathsf{Sim}_{\mathsf{csf}}$ of Π on input $(1^\lambda, 1^Q, 1^s)$.

   For the i^{th} circuit query C_i made by $\mathcal{A}$, do the following: compute the simulated circuit encodings $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{Sim}_{\mathsf{csf}}(C_i)$. Let $\mathsf{E}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \cdot)$. For every $\boldsymbol{u} \in [N]$, $i \in [Q_1]$,

   - If $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}} \cap \mathbf{S}_i$, generate a functional key $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{1fe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, \mathsf{E}_i^{\boldsymbol{u}})$.
   - If $\boldsymbol{u} \notin \mathcal{S}_{\mathsf{corr}}$ and $\boldsymbol{u} \in \mathbf{S}_i$, generate a simulated functional key for $\mathsf{E}_i^{\boldsymbol{u}}$. That is, compute $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{sim}_{\boldsymbol{u}}(\mathsf{E}_i^{\boldsymbol{u}})$.

   Set $\mathsf{SK}_{C_i} = \left(\mathbf{S}_i,\ \left\{\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}}\right\}_{\boldsymbol{u} \in \mathbf{S}_i}\right)$. Send SK_{C_i} to $\mathcal{A}$.

5. $\mathcal{A}$ outputs the challenge message x. Let $\mathcal{V} = \{(C_i, C_i(x))\ :\ \forall i \in [Q_1]\}$.

6. Next step is to generate the challenge ciphertext. Compute $\mathsf{Sim}_{\mathsf{csf}}(1^{|x|}, \mathcal{S}_{\mathsf{corr}}, \mathcal{V})$ to obtain $\{\widehat{x}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}}$ and evaluations $\{\widehat{y}_i^{\boldsymbol{u}}\}_{i \in [Q_1], \boldsymbol{u} \in \mathbf{S}_i}$ such that $\widehat{y}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}})$ for $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}$. For every $\boldsymbol{u} \in [N]$,

   - If $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}$, compute $\mathsf{ct}_{\boldsymbol{u}} \leftarrow \mathsf{1fe.Enc}(\mathsf{mpk}_{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}})$.
   - If $\boldsymbol{u} \notin \mathcal{S}_{\mathsf{corr}}$: let $V_{\boldsymbol{u}} = \{\widehat{y}_i^{\boldsymbol{u}}\ :\ \forall i \in [Q_1]\}$ for every $\boldsymbol{u} \in [N]$. Compute the simulated ciphertext $\mathsf{ct}_{\boldsymbol{u}} \leftarrow \mathsf{sim}_{\boldsymbol{u}}(1^{|x|}, V_{\boldsymbol{u}})$.

   Set $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$. Send CT to $\mathcal{A}$.

7. In the next phase, $\mathcal{A}$ makes $Q - Q_1$ circuit queries $C_{Q_1+1}, \ldots, C_Q$.

   For $i = Q_1{+}1, \ldots, Q$, compute the simulator $\mathsf{Sim}_{\mathsf{csf}}(i, C_i, C_i(x))$ to obtain the simulated circuit encoding $\widehat{C_i}^{\boldsymbol{u}}$ and simulated values $\{\widehat{y}_i^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathbf{S}_i}$. Let $\mathsf{E}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \cdot)$. For every $\boldsymbol{u} \in [N]$, for every $i = Q_1 + 1, \ldots, Q$,

   - If $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}} \cap \mathbf{S}_i$, generate a functional key $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{1fe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, \mathsf{E}_i^{\boldsymbol{u}})$.
   - If $\boldsymbol{u} \notin \mathcal{S}_{\mathsf{corr}}$ and $\boldsymbol{u} \in \mathbf{S}_i$, generate a simulated functional key for $\mathsf{E}_i^{\boldsymbol{u}}$; that is, compute $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{sim}_{\boldsymbol{u}}(\mathsf{E}_i^{\boldsymbol{u}}, \widehat{y}_i^{\boldsymbol{u}})$.

   Set $\mathsf{SK}_{C_i} = \left(\mathbf{S}_i,\ \left\{\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}}\right\}_{\boldsymbol{u} \in \mathbf{S}_i}\right)$. Send SK_{C_i} to $\mathcal{A}$.

8. $\mathcal{A}$ outputs b. Output b.

We describe the hybrids below.

<u>**Hyb**$_1$</u>: This hybrid consists of the following steps.

1. Ch receives the query bound Q and the maximum circuit size s from $\mathcal{A}$.

2. Sample Q sets as follows: for every $i \in [N]$, sample a set $\mathbf{S}_i$ of size n uniformly at random from $[N]$.

3. Ch computes the master public key mpk. For every $\boldsymbol{u} \in [N]$, Ch generates $(\mathsf{mpk}_{\boldsymbol{u}}, \mathsf{msk}_{\boldsymbol{u}}) \leftarrow \mathsf{1fe.Setup}(1^\lambda, 1^Q, 1^s)$. Ch sends $\mathsf{MPK} = (\mathsf{mpk}_1, \ldots, \mathsf{mpk}_N)$ to $\mathcal{A}$.

4. $\mathcal{A}$ makes Q_1 circuit queries $C_1, \ldots, C_{Q_1}$ to Ch. For every $i \in [Q_1]$, the challenger computes the functional key of C_i as follows:

   - Compute $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C_i)$.
   - Let $\mathsf{E}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \cdot)$, for every $\boldsymbol{u} \in \mathbf{S}_i$. Generate a functional key for $\mathsf{E}_i^{\boldsymbol{u}}$; that is, compute $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{1fe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, \mathsf{E}_i^{\boldsymbol{u}})$ for every $\boldsymbol{u} \in \mathbf{S}_i$.

   Set $\mathsf{SK}_{C_i} = \left(\mathbf{S}_i, \ \left\{\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}}\right\}_{\boldsymbol{u} \in \mathbf{S}_i}\right)$. Send SK_{C_i} to $\mathcal{A}$.

5. $\mathcal{A}$ submits the challenge message x. Ch does the following:

   - Compute $(\widehat{x}^1, \ldots, \widehat{x}^N) \leftarrow \mathsf{InpEnc}\left(1^\lambda, 1^Q, 1^s, x\right)$.
   - For every $\boldsymbol{u} \in [N]$, compute $\mathsf{ct}_i \leftarrow \mathsf{FE.Enc}\left(\mathsf{mpk}_{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}}\right)$.

   Set $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$. Send CT to $\mathcal{A}$.

6. $\mathcal{A}$ makes $Q - Q_1$ circuit queries $C_{Q_1+1}, \ldots, C_Q$ to Ch. For every $i = Q_1 + 1, \ldots, Q$, Ch computes the functional key of C_i as follows:

   - Compute $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C_i)$.
   - Let $\mathsf{E}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \cdot)$, for every $\boldsymbol{u} \in \mathbf{S}_i$. Generate a functional key for $\mathsf{E}_i^{\boldsymbol{u}}$; that is, compute $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{1fe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, \mathsf{E}_i^{\boldsymbol{u}})$ for every $\boldsymbol{u} \in \mathbf{S}_i$.

   Set $\mathsf{SK}_{C_i} = \left(\mathbf{S}_i, \ \left\{\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}}\right\}_{\boldsymbol{u} \in \mathbf{S}_i}\right)$. Send SK_{C_i} to $\mathcal{A}$.

7. $\mathcal{A}$ outputs bit b. Otherwise, output b.

Note that the experiment described above is phrased differently from the real experiment of the bounded-key FE scheme. In the real experiment of the bounded-key FE scheme, the sets $\{\mathbf{S}_i\}$ are sampled as part of the key generation algorithm but in the above experiment the sets are sampled in the beginning of the experiment. However, these two different formulations lead to the same distribution of the output of the experiment.

<u>$\mathbf{Hyb}_2$</u>: If the number of pairwise intersections of the sets $\{\mathbf{S}_i\}$ is more than the threshold t, the experiment aborts. More formally, we change the second bullet in the above hybrid.

2. Sample Q sets as follows: for every $i \in [N]$, sample $\mathbf{S}_i \xleftarrow{\$} [N]$ of size n. Construct a set $\mathcal{S}_{\mathsf{corr}}$ as follows: include an index $\boldsymbol{u} \in [N]$ in the set $\mathcal{S}_{\mathsf{corr}}$ if and only if $\boldsymbol{u} \in \mathbf{S}_i \cap \mathbf{S}_j$, for some $i, j \in [Q]$ and $i \neq j$. If $|\mathcal{S}_{\mathsf{corr}}| > t$, output $\perp$.

Claim 11. *The statistical distance between the output distributions of* $\mathbf{Hyb}_1$ *and* $\mathbf{Hyb}_2$ *is negligible in security parameter* λ.

Proof. This follows from the lemma stated and proven in [24]. Note that $\left|\bigcup_{i_1 \neq i_2} (\mathbf{S}_{i_1} \cap \mathbf{S}_{i_2})\right| = \mathcal{S}_{\mathsf{corr}}$.

Lemma 5 (Small Pairwise Intersection Lemma [24]). *Let $Q \in \mathbb{Z}_{>0}$. Set $t = \Theta(Q^2\lambda), N = \Theta(Q^2t^2)$ and $n = \Theta(t)$. Then,*

$$\mathsf{Pr}\left[\left|\bigcup_{i_1 \neq i_2} (\mathbf{S}_{i_1} \cap \mathbf{S}_{i_2})\right| > t\right] \leq \mathsf{negl}(\lambda),$$

for some negligible function negl.

□

<u>$\mathbf{Hyb}_{3.\mathbf{j}}$, for every $\mathbf{j} \in [N]$</u>: The first $\mathbf{j} - 1$ instantiations of 1fe are simulated while the rest of the instantiations are invoked as in $\mathbf{Hyb}_2$. We consider the following hybrid experiment.

1. Ch receives the query bound Q and the maximum circuit size s from $\mathcal{A}$.

2. For every $i \in [N]$, sample a set $\mathbf{S}_i$ of size n uniformly at random from $[N]$. Construct a set $\mathcal{S}_{\mathsf{corr}}$ as follows: include an index $\boldsymbol{u}$ in the set $\mathcal{S}_{\mathsf{corr}}$ if and only if $\boldsymbol{u} \in \mathbf{S}_i \cap \mathbf{S}_j$, for some $i, j \in [Q]$ and $i \neq j$. If $|\mathcal{S}_{\mathsf{corr}}| > t$, output $\perp$.

3. Ch computes the master public key mpk. For every $\boldsymbol{u} \in [N]$,

   - If $\boldsymbol{u} \geq \mathbf{j}$ or $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}$: Ch generates $(\mathsf{mpk}_{\boldsymbol{u}}, \mathsf{msk}_{\boldsymbol{u}}) \leftarrow \mathsf{1fe.Setup}(1^\lambda, 1^Q, 1^s)$.
   - Otherwise: $\mathsf{mpk}_{\boldsymbol{u}} \leftarrow \mathsf{sim}_{\boldsymbol{u}}(1^\lambda, 1^s)$.

   Ch sends $\mathsf{MPK} = (\mathsf{mpk}_1, \ldots, \mathsf{mpk}_N)$ to $\mathcal{A}$.

4. $\mathcal{A}$ makes Q_1 circuit queries $C_1, \ldots, C_{Q_1}$ to Ch. For every $i \in [Q_1]$, the challenger computes the functional key of C_i as follows: for every $\boldsymbol{u} \in \mathbf{S}_i$,

   - If $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}} \cap \mathbf{S}_i$ or $\boldsymbol{u} \geq \mathbf{j}$: compute $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C_i)$. Let $\mathsf{E}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \cdot)$. Generate a functional key for $\mathsf{E}_i^{\boldsymbol{u}}$; that is, compute $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{1fe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, \mathsf{E}_i^{\boldsymbol{u}})$.
   - Otherwise if $\boldsymbol{u} \in \mathbf{S}_i \backslash \mathcal{S}_{\mathsf{corr}}$ and $\boldsymbol{u} < \mathbf{j}$: compute the simulated circuit encodings $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{Sim}_{\mathsf{csf}}(C_i)$. Let $\mathsf{E}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \cdot)$. For every $\boldsymbol{u} \in [N]$, generate a simulated functional key for $\mathsf{E}_i^{\boldsymbol{u}}$. That is, compute $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{sim}_{\boldsymbol{u}}(\mathsf{E}_i^{\boldsymbol{u}})$.

   Set $\mathsf{SK}_{C_i} = \left(\mathbf{S}_i, \ \{\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}}\}_{\boldsymbol{u} \in \mathbf{S}_i}\right)$. Send SK_{C_i} to $\mathcal{A}$.

5. $\mathcal{A}$ submits the challenge message x. Ch does the following:

   - Compute $(\widehat{x}^1, \ldots, \widehat{x}^N) \leftarrow \mathsf{InpEnc}\left(1^\lambda, 1^Q, 1^s, x\right)$.
   - If $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}$ or $\boldsymbol{u} \geq \mathbf{j}$: compute $\mathsf{ct}_{\boldsymbol{u}} \leftarrow \mathsf{FE.Enc}\left(\mathsf{mpk}_{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}}\right)$.
   - If $\boldsymbol{u} \notin \mathcal{S}_{\mathsf{corr}}$ and $\boldsymbol{u} < \mathbf{j}$: for every $i \in [Q_1]$, let $\widehat{y}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}})$ and let $V_{\boldsymbol{u}} = \{\widehat{y}_i^{\boldsymbol{u}} : \forall i \in [Q_1]\}$. Compute the simulated ciphertext $\mathsf{ct}_{\boldsymbol{u}} \leftarrow \mathsf{sim}(1^{|x|}, V_{\boldsymbol{u}})$.
   - Otherwise: compute $\mathsf{ct}_{\boldsymbol{u}} \leftarrow \mathsf{FE.Enc}\left(\mathsf{mpk}_{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}}\right)$.

   Set $\mathsf{CT} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$. Send CT to $\mathcal{A}$.

6. $\mathcal{A}$ makes $Q - Q_1$ circuit queries $C_{Q_1+1}, \ldots, C_Q$ to Ch. For every $i = Q_1 + 1, \ldots, Q$, Ch computes the functional key of C_i as follows:

   - If $\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}} \cap \mathbf{S}_i$ or $\boldsymbol{u} \geq \mathbf{j}$: compute $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C_i)$. Let $\mathsf{E}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \cdot)$, for every $\boldsymbol{u} \in \mathbf{S}_i$. Generate a functional key for $\mathsf{E}_i^{\boldsymbol{u}}$; that is, compute $\mathsf{sk}^{\boldsymbol{u}}_{\mathsf{E}_i} \leftarrow \mathsf{1fe.KeyGen}(\mathsf{msk}_{\boldsymbol{u}}, \mathsf{E}_i^{\boldsymbol{u}})$ for every $\boldsymbol{u} \in \mathbf{S}_i$.
   - Otherwise if $\boldsymbol{u} \in \mathbf{S}_i \backslash \mathcal{S}_{\mathsf{corr}}$ and $\boldsymbol{u} < \mathbf{j}$: compute $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^s, C_i)$. Let $\mathsf{E}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \cdot)$. For every $\boldsymbol{u} \in [N]$, generate a simulated functional key for $\mathsf{E}_i^{\boldsymbol{u}}$. That is, compute $\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}} \leftarrow \mathsf{sim}_{\boldsymbol{u}}(\mathsf{E}_i^{\boldsymbol{u}}, \widehat{y}_i^{\boldsymbol{u}})$, where $\widehat{y}_i^{\boldsymbol{u}} = \mathsf{Local}(\widehat{C_i}^{\boldsymbol{u}}, \widehat{x}^{\boldsymbol{u}})$.

   Set $\mathsf{SK}_{C_i} = \left(\mathbf{S}_i, \ \{\mathsf{sk}_{\mathsf{E}_i^{\boldsymbol{u}}}\}_{\boldsymbol{u} \in \mathbf{S}_i}\right)$. Send SK_{C_i} to $\mathcal{A}$.

7. $\mathcal{A}$ outputs bit b. Otherwise, output b.

The following claim immediately follows from the security of $\mathsf{1fe}$.

Claim 12. *For every* $\mathbf{j} \in [N-1]$, *assuming that* $\mathsf{1fe}$ *is secure, it holds that output distributions of* $\mathbf{Hyb}_{3.\mathbf{j}}$ *and* $\mathbf{Hyb}_{3.\mathbf{j}+1}$ *are computationally indistinguishable.*

$\underline{\mathbf{Hyb}_4}$: Recall that in the previous hybrid, the input encodings $(\widehat{x}^1, \ldots, \widehat{x}^N)$ are generated by executing $\mathsf{InpEnc}(1^\lambda, 1^Q, 1^s, x)$. Instead, in this hybrid, compute $\mathsf{Sim}_{\mathsf{csf}}(1^{|x|}, \mathcal{S}_{\mathsf{corr}}, \mathcal{V})$, where $\mathcal{V} = \{(C_i, C_i(x)) : \forall i \in [Q_1]\}$ to obtain $\{\widehat{x}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}}$. Thus, using $\{\widehat{x}^{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}}$, $\mathsf{1fe}$ ciphertexts $\{\mathsf{ct}_{\boldsymbol{u}}\}_{\boldsymbol{u} \in \mathcal{S}_{\mathsf{corr}}}$ can be generated as in the previous hybrid. And the rest of $\mathsf{1fe}$ ciphertexts $\{\mathsf{ct}_{\boldsymbol{u}}\}_{\boldsymbol{u} \notin \mathcal{S}_{\mathsf{corr}}}$ are simulated using the simulator of $\mathsf{1fe}$.

For every circuit C_i queried by the adversary, where $i \in [Q]$, generate $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{Sim}_{\mathsf{csf}}(C_i)$ if $i \leq Q_1$, otherwise generate $\left(\widehat{C_i}^1, \ldots, \widehat{C_i}^N\right) \leftarrow \mathsf{Sim}_{\mathsf{csf}}(C_i, C_i(x))$.

The following claim immediately follows from the security of Π.

Claim 13. *Assuming that* Π *is secure, it holds that the output distributions of* $\mathbf{Hyb}_{3.N}$ *and* $\mathbf{Hyb}_4$ *are computationally indistinguishable.*

$\underline{\mathbf{Hyb}_5}$: This corresponds to the simulated experiment.

Claim 14. *The output distributions of* $\mathbf{Hyb}_3$ *and* $\mathbf{Hyb}_4$ *are identically distributed.*

$\square$

Instantiation. The bounded-key functional encryption scheme described above makes black box usage of $\mathsf{InpEnc}(\cdot)$ algorithm of Π. Moreover, in the construction of Π described in the proof of lemma 4, pseudorandom generators are only used in $\mathsf{InpEnc}(\cdot)$. Furthermore, $\mathsf{InpEnc}(\cdot)$ only makes black box calls to the pseudorandom generator. Thus, the bounded-key functional encryption scheme described above, when instantiated with Π described in the proof of lemma 4, yields a bounded-key scheme that makes only oracle calls to cryptographic primitives (public-key encryption and pseudorandom generators).

References

[1] Shweta Agrawal. Stronger security for reusable garbled circuits, general definitions and attacks. In *CRYPTO*, 2017.

[2] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In *TCC*, pages 173–205, 2017.

[3] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In *Annual International Cryptology Conference*, pages 395–424. Springer, 2018.

[4] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, 2015.

[5] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptology ePrint Archive*, 2015:730, 2015.

[6] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Perfect secure computation in two rounds. In *Theory of Cryptography Conference*, pages 152–174. Springer, 2018.

[7] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[8] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *STOC*, 1990.

[9] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, 1988.

[10] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 500–532. Springer, 2018.

[11] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 171–190, 2015.

[12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography*, pages 253–273. Springer, 2011.

[13] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. In *EUROCRYPT*, pages 535–564, 2018.

[14] Yilei Chen, Vinod Vaikuntanathan, Brent Waters, Hoeteck Wee, and Daniel Wichs. Traitor-tracing from lwe made simple and attribute-based. Cryptology ePrint Archive, Report 2018/897, 2018. https://eprint.iacr.org/2018/897.

[15] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Bounded cca2-secure encryption. In *ASIACRYPT*, 2007.

[16] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In *EUROCRYPT*, pages 65–82, 2002.

[17] Nico Döttling and Sanjam Garg. From selective IBE to full IBE and selective HIBE. In *TCC*, pages 372–408, 2017.

[18] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *CRYPTO*, pages 537–569, 2017.

[19] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round mpc: information-theoretic and black-box. In *Theory of Cryptography Conference*, pages 123–151. Springer, 2018.

[20] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In *TCC*, pages 419–442, 2016.

[21] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 468–499. Springer, 2018.

[22] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.

[23] Shafi Goldwasser, Allison B. Lewko, and David A. Wilson. Bounded-collusion IBE from key homomorphism. In *TCC*, 2012.

[24] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.

[25] Swee-Huay Heng and Kaoru Kurosawa. k-resilient identity-based encryption in the standard model. In *CT-RSA*, pages 67–80, 2004.

[26] Gene Itkis, Emily Shen, Mayank Varia, David Wilson, and Arkady Yerukhimovich. Bounded-collusion attribute-based encryption from minimal assumptions. In *PKC*, pages 67–87, 2017.

[27] Baiyu Li and Daniele Micciancio. Compactness vs collusion resistance in functional encryption. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 443–468, 2016.

[28] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 463–472. ACM, 2010.

[29] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.

[30] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

[31] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.