

Combiners for Functional Encryption, Unconditionally

Aayush Jain
UCLA

aayushjain@cs.ucla.edu

Nathan Manohar
UCLA

nmanohar@cs.ucla.edu

Amit Sahai
UCLA

sahai@cs.ucla.edu

Abstract

Functional encryption (FE) combiners allow one to combine many candidates for a functional encryption scheme, possibly based on different computational assumptions, into another functional encryption candidate with the guarantee that the resulting candidate is secure as long as at least one of the original candidates is secure. The fundamental question in this area is whether FE combiners exist. There have been a series of works (Ananth et. al. (CRYPTO '16), Ananth-Jain-Sahai (EUROCRYPT '17), Ananth et. al (TCC '19)) on constructing FE combiners from various assumptions.

We give the first *unconditional* construction of combiners for functional encryption, resolving this question completely. Our construction immediately implies an unconditional universal functional encryption scheme, an FE scheme that is secure if such an FE scheme exists. Previously such results either relied on algebraic assumptions or required subexponential security assumptions.

1 Introduction

In cryptography, many interesting cryptographic primitives rely on computational assumptions. Over the years, many assumptions have been proposed such as factoring, quadratic residuosity, decisional Diffie-Hellman, learning with errors, and many more. However, despite years of research, the security of these assumptions is still not firmly established. Indeed, we do not even know how to prove $\mathbf{P} \neq \mathbf{NP}$; our understanding of algebraic hardness is even more speculative. Moreover, we also do not have a strong understanding of how different cryptographic assumptions compare against each other. For instance, it is not known whether decisional Diffie-Hellman is a weaker or a stronger assumption than learning with errors. This inability to adequately compare different cryptographic assumptions induces the following problematic situation: suppose we have a cryptographic primitive (say, public key encryption) with many candidate constructions based on a variety of assumptions, and we want to pick the most secure candidate to use. Unfortunately, due to our limited knowledge of how these assumptions compare against each other, we cannot determine which candidate is the most secure one.

Unconditional Cryptographic Combiners. Cryptographic combiners were introduced to handle the above issue. Given many candidates of a cryptographic primitive, possibly based on different assumptions, a cryptographic combiner takes these candidates and produces another candidate for the same primitive with the guarantee that this new candidate is secure as long as at least *one* of the original candidates is secure. For example, a combiner for public key encryption can be used to transform two candidates, one based on decisional Diffie-Hellman and the other on learning

with errors, into a new public-key encryption candidate that is secure provided *either* decisional Diffie-Hellman *or* learning with errors is secure. Thus, this new public-key encryption candidate relies on a strictly weaker assumption than the original two candidate constructions and allows us to hedge our bets on the security of the two original assumptions.

Furthermore, even if an underlying primitive, such as public-key encryption, requires an unproven hardness assumption, the security of a *combiner* for that primitive can be unconditional. Therefore, cryptographic combiners stand out in the world of cryptography in the sense that they are one of the few useful cryptographic objects that do not inherently require reliance on hardness assumptions. And indeed, combiners for fundamental primitives like one-way functions, public-key encryption, and oblivious transfer are known to exist unconditionally [HKN⁺05, FL07, HIKN08, IPS08].

Obtaining unconditional combiners is particularly important because the entire purpose of constructing combiners is to make cryptographic constructions *future-proof* in case assumptions break down. In this work, we study combiners for *functional encryption*, an area where studying combiners is particularly important and where, prior to our work, only conditional constructions were known [AJN⁺16, AJS17, ABJ⁺19] (and in fact, these previous results required either algebraic or sub-exponentially strong assumptions). We obtain the first unconditional combiner for functional encryption. Furthermore, we do so by providing a general compiler, significantly simplifying previous work in this area. Along the way, we define and provide constructions of input-local MPC protocols, input-local garbling schemes, and combiner-friendly homomorphic secret sharing schemes, primitives that may be of independent interest.

Combiners for Functional Encryption. Functional encryption (FE), introduced by [SW05] and first formalized by [BSW11, O’N10], is one of the core primitives in the area of computing on encrypted data. This notion allows an authority to generate and distribute constrained keys associated with functions f_1, \dots, f_q , called *functional keys*, which can be used to learn the values $f_1(x), \dots, f_q(x)$ given an encryption of x . Intuitively, the security notion states that the functional keys associated with f_1, \dots, f_q and an encryption of x reveal nothing beyond the values $f_1(x), \dots, f_q(x)$.

Function encryption has opened the floodgates to important cryptographic applications that have long remained elusive. These applications include, but are not limited to, multi-party non-interactive key exchange [GPSZ17], universal samplers [GPSZ17], reusable garbled circuits [GKP⁺13], verifiable random functions [GHKW17, Bit17, BGJS17], and adaptive garbling [HJO⁺16]. FE has also helped improve our understanding of important theoretical questions, such as the hardness of Nash equilibrium [GPS16, GPSZ17]. One of the most important applications of FE is its implication to indistinguishability obfuscation (iO for short) [AJ15, BV15]. In fact, if we are willing to tolerate subexponential security loss, then even secret-key FE is enough to imply iO [BNPW16, KS17, KNT18].

Over the past few years, many constructions of functional encryption have been proposed [GGH⁺13, GGHZ14, Lin16, LV16, AS17, Lin17, LT17, AJS18, Agr19, LM18, AJL⁺19] and studying what assumptions suffice for constructing general-purpose FE remains a very important and active area of investigation. Recent cryptanalytic attacks [CHL⁺15, HJ16, CLLT16, CJL16, CGH⁺15, BBKK18, LV16, BHJ⁺19] on FE schemes further highlight the importance of careful study. Given these results, we should hope to minimize the trust we place on any individual FE candidate.

The notion of a functional encryption combiner achieves this purpose. Informally speaking,

a functional encryption combiner allows for combining many functional encryption candidates in such a way that the resulting FE candidate is secure as long as at least *one* of the initial FE candidates is secure. In other words, a functional encryption combiner says that it suffices to place trust collectively on multiple FE candidates, instead of placing trust on any specific FE candidate. Furthermore, FE combiners are an important area of study for the following reasons:

- Most importantly, it gives a mechanism to hedge our bets and distribute our trust over multiple constructions. This has been highlighted above.
- Often, constructions of FE combiners give rise to constructions of robust FE combiners generically [AJS17, ABJ⁺19]. Any robust FE combiner gives us a universal construction of FE, which is an explicit FE scheme that is secure as long as there exists a secure functional encryption scheme.
- Studying FE combiners helps improve our understanding of the nature of assumptions we need to build FE.
- They give rise to theoretically important results in other branches of cryptography, such as round-optimal low-communication MPC [ABJ⁺19].
- Constructions of robust FE combiners have encouraged research on understanding correctness amplification for FE, iO [BV16, AJS17], and other fundamental cryptographic primitives [BV17].
- Finally, due to connections to security amplification, techniques used to build FE combiners are useful to give better constructions of FE. In particular, the work of [AJS18] used techniques developed from the study of FE combiners to provide a generic security amplification of FE, which proved pivotal in giving the first construction of FE that does not rely on multilinear maps and makes use of simply stated, instance-independent assumptions.

There have been a series of works in this area. The starting point was the work of two concurrent papers [AJN⁺16, FHNS16], both appearing at CRYPTO, that studied the related question of obfuscation combiners. This was followed up by the work of [AJS17], which gave a construction of FE combiners (and universal FE) assuming the existence of a subexponentially secure FE algorithm. They also gave a construction of a robust FE combiner assuming LWE. Then [ABJ⁺19] gave construction of a robust FE combiner (and universal obfuscation) relying on the algebraic assumption of the existence of constant degree randomizing polynomials (which are known to exist assuming number-theoretic assumptions such as LWE, DDH, and quadratic residuosity). However, until now, the ultimate question in this area, of whether FE combiners exist without making any additional assumptions, has remained open.

1.1 Our Contributions

In this paper, we consider the following questions.

What is the minimal assumption necessary to construct FE combiners and universal FE?

In particular,

Is it possible to construct FE combiners and universal FE unconditionally?

We resolve the above question in the affirmative and prove the following.

Theorem 1 (Informal). *There exists an unconditionally secure FE combiner for P/poly.*

It turns out that our construction of an FE combiner also gives rise to a robust FE combiner using the results of [AJS17, ABJ⁺19].

Corollary 1 (Informal). *There exists an unconditionally secure robust FE combiner for P/poly.*

As any robust FE combiner gives a universal FE scheme [AJN⁺16, AJS17], we obtain the following additional result.

Corollary 2 (Informal). *There exists an unconditional construction of a universal FE scheme for P/poly.*

We note that, as was the case in previous constructions, our construction of a universal FE scheme is parameterized by the maximum run-time of any of the algorithms of the secure FE scheme.

1.2 Technical Overview

Our starting point is the observation that FE combiners are related to the notion of secure multi-party computation and function secret sharing (also known as homomorphic secret sharing [BGI15, BGI16a, BGI16b, MW16, BGG⁺18]). Suppose for a function f , it was possible to give out function shares f_1, \dots, f_n such that for any input x , we can n -out-of- n secret share x into shares x_1, \dots, x_n and recover $f(x)$ given $f_1(x_1), \dots, f_n(x_n)$. Then, we would be able to build an FE combiner in the following manner. Given an input x , the encryptor would n -out-of- n secret share x and encrypt the i th share x_i under the i th FE candidate FE_i (depicted in Figure 1). To generate a function key for a function f , FE_i would generate a function key for function share f_i . Using these ciphertexts and function keys, it would be possible to recover $f_i(x_i)$, from which it would be possible to recover $f(x)$. Security would follow from the fact that since at least one FE candidate is secure, one of the input shares remains hidden, hiding the input. This overall approach was used in [AJS17, ABJ⁺19] to construct FE combiners from LWE. In this work, we would like to minimize the assumptions needed to construct an FE combiner, and, unfortunately, we do not know how to construct such a function sharing scheme for polynomial-sized circuits from one-way functions. Note that since FE implies one-way functions, any FE combiner can assume the existence of one-way functions since the individual one-way function candidates arising from each FE candidate can be trivially combined by independent concatenation (direct product) of the candidate one-way functions.

Our first step towards constructing an FE combiner unconditionally is that we observe that it is easy to build an FE combiner for a *constant* number of FE candidates by simply nesting the candidates. For example, if we had 2 FE candidates, FE_1 and FE_2 , we could combine these two candidates by simply having encryption encrypt first under FE_1 and then encrypt the resulting ciphertext under FE_2 . To generate a secret key for a function f , we would generate a function key $\text{SK}_{f,1}$ for f under FE_1 and then generate a function key $\text{SK}_{f,2}$ for the function that runs $\text{FE}_1.\text{Dec}(\text{SK}_{f,1}, \cdot)$ under FE_2 . The function key $\text{SK}_{f,2}$ would then be the function key for f under the combined FE scheme. Using nestings of candidates, we can replace our original FE candidates with these new nested candidates. For example, if we use 2-nestings, we can consider all possible

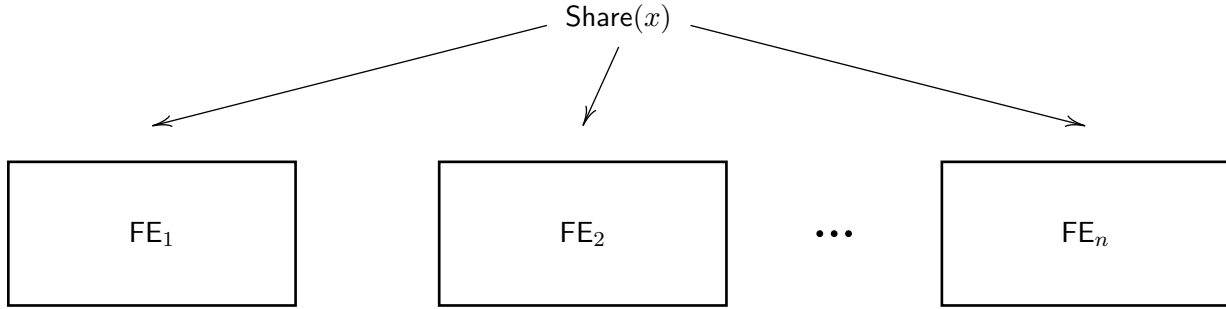


Figure 1: A pictorial overview of splitting x amongst n FE candidates.

2-nestings $FE_{i,j}$ for $i, j \in [n]$ as our new set of FE candidates. Observe now that we have replaced our original n FE candidates with n^2 “new” FE candidates. At first glance, this appears to not have helped much. However, note that previously, we needed to consider function sharing schemes that were secure against up to $n - 1$ corruptions. When using nested candidates, it follows that if FE_{i^*} was originally secure, then $FE_{i,j}$ with at least one of $i, j = i^*$ is also secure. We show how to leverage this new corruption pattern of the candidates in the following manner.

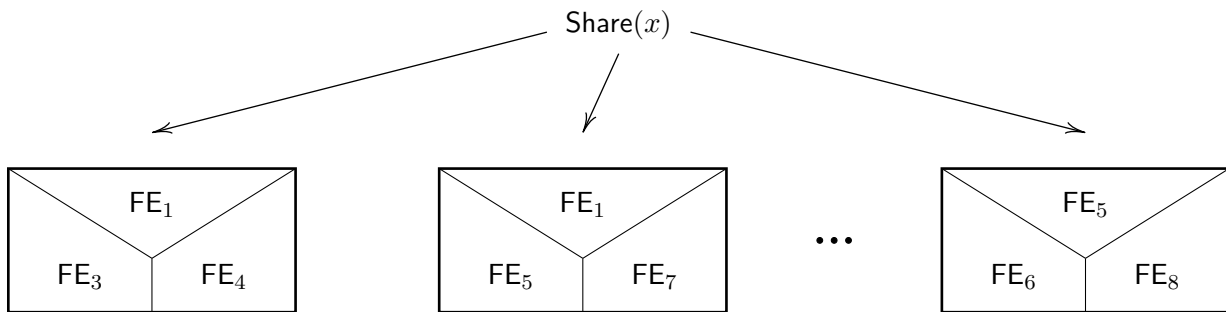


Figure 2: A pictorial overview of 3-nested FE candidates (the required level of nesting in our construction). If FE_5 is secure, then $FE_{1,5,7}$ and $FE_{5,6,8}$ are secure.

Suppose we had a “special” MPC protocol Φ where every bit in the transcript of an execution of Φ can be computed by a function on the inputs (and random coins) of a *constant* number of parties (say 2). Furthermore, the output of Φ can be determined solely from the transcript and Φ is secure against a semi-honest adversary that corrupts up to $n - 1$ parties. If Φ has the above properties, then the transcript of an execution of Φ can be determined via an alternate computation. Instead of running Φ normally to obtain the transcript, we can instead compute jointly on all pairs of parties’ inputs (and randomness) to obtain the transcript. That is, if a bit τ_α in the transcript τ can be computed given only the inputs (and randomness) of parties P_i and P_j (we say it “depends” on parties P_i and P_j), then we can determine the value of τ_α in an execution of Φ by computing this function on (x_i, r_i) and (x_j, r_j) (the inputs and randomness of these two parties) rather than executing the protocol in the normal fashion. Proceeding in the same manner for every bit in the transcript, we can obtain the same exact transcript that we would have by executing the protocol normally, but we are able to do so by only evaluating functions on two parties’ inputs (and randomness).

This observation leads us to the following approach for constructing an FE combiner. To encrypt an input x , additively secret share x into n shares (x_1, \dots, x_n) and encrypt each pair of shares (x_i, x_j) under $\text{FE}_{i,j}$. To generate a function key for a function f , consider the MPC protocol that computes $f(x_1 \oplus \dots \oplus x_n)$. Then, for every bit τ_α in the transcript of such a protocol, if τ_α “depends” on parties P_i, P_j , we would generate a function key under $\text{FE}_{i,j}$ for the circuit C_{τ_α} that computes τ_α given x_i, x_j .

This approach immediately runs into the following problem. The MPC protocol is randomized, whereas the function keys in an FE scheme are for *deterministic* functions. Moreover, an FE ciphertext needs to be compatible with many function keys. Fortunately, these problems can easily be solved by having the encryptor also generate a PRF key K_i for each party P_i . The encryptor now encrypts (x_i, x_j, K_i, K_j) under FE candidate $\text{FE}_{i,j}$ and uses K_i and some fixed tag tag_f embedded in the function key for f to generate the randomness of P_i in the evaluation of the MPC protocol. Now, by using the function keys for the C_{τ_α} ’s, it is possible for the decryptor to recover all the bits in the transcript of an execution of the protocol and, therefore, recover $f(x)$. Security would follow from the fact that if candidate FE_{i^*} is secure, then x_{i^*} and K_{i^*} remain hidden, and we can use the security of the MPC protocol to simulate the view of party P_{i^*} .

If such an MPC protocol as described above could be found, the above would suffice for constructing an FE combiner. However, the goal of this work is to construct an FE combiner *unconditionally* and so we would like to only assume the existence of one-way functions. However, semi-honest MPC secure against up to $n - 1$ corruptions requires oblivious transfer (OT), which we do not want to assume. To deal with this, we adapt our MPC idea to settings with correlated randomness, such as the OT-hybrid model.

A first attempt at adapting this idea to protocols in the OT-hybrid model is the following. Suppose that we have a “special” MPC protocol Φ where every bit in the transcript of an execution of Φ can be computed by a function on the inputs (and random coins/correlated randomness) of a *constant* number of parties (say 2). Furthermore, the output of Φ can be determined solely from the transcript and Φ is secure against a semi-honest adversary that corrupts up to $n - 1$ parties in the OT-hybrid model.

The first challenge is to instantiate the OT oracle. This can be done by having shared PRF keys $K_{i,j}$ between all pairs of parties P_i and P_j . Then $K_{i,j}$ will be used to generate correlated randomness between P_i and P_j . We can generate all the correlated randomness prior to the protocol execution and include it as part of the input to a party P_i . This allows us to generate correlated randomness, but we still run into a second issue. Since a party P_i has correlated randomness between itself and all other parties, its input now depends on all other parties! So, it appears that constant nestings of FE candidates will no longer suffice.

Fortunately, this second issue can be mitigated by a more refined condition on the “special” MPC protocol Φ . Let $(r_{i,j}, r_{j,i})$ denote the correlated randomness pair between parties P_i and P_j , where $r_{i,j}$ and $r_{j,i}$ are given to P_i and P_j , respectively. Instead of having the functions that compute bits of the transcript of Φ take as input the entire correlated randomness string $\{r_{i,j}\}_{j \neq i \in [n]}$ held by a party P_i , we instead allow it to take single components $r_{i,j}$ as input. If the function takes as input $r_{i,j}$, then *both* parties P_i and P_j are counted in the number of parties that the function depends on. More formally, the condition on the “special” MPC protocol Φ becomes the following. Let (x_i, r_i) denote the input and randomness of a party P_i and let $r_{i,j}$ denote the correlated randomness between parties P_i and P_j held by P_i . Every bit τ_α in the transcript τ of an execution of Φ can be

computed by some deterministic function f_α on input

$$((x_i)_{i \in \mathcal{S}_\alpha}, (r_i)_{i \in \mathcal{S}_\alpha}, (r_{i,j})_{i,j \in \mathcal{S}_\alpha}),$$

where $|\mathcal{S}_\alpha| \leq t$ for some constant t . We call such an MPC protocol a t -input-local MPC protocol and define this formally in [Section 4](#).

To summarize, if we had a t -input-local MPC protocol for some constant t , then we would be able to construct an FE combiner unconditionally using the ideas detailed above. However, it is unclear how to construct such an MPC protocol, and, unfortunately, no protocol in the literature for all polynomial-sized circuits in the OT-hybrid model satisfies all our required properties. However, the 2-round semi-honest MPC protocol of Garg-Srinivasan [\[GS18\]](#) transformed to operate in the OT-hybrid model [\[GIS18\]](#) comes close. At a high level, this is because they compile an MPC protocol into a series of garbled circuits, where each garbled circuit is computed by a single party. However, there are several bottlenecks that make their protocol initially incompatible with our schema. One observation is that the protocol of [\[GS18, GIS18\]](#) contains a pre-processing phase that causes the initial state (effectively input) of each party to be dependent on all other parties. This might seem like a major issue since messages dependent only on a single parties' state can now depend on all parties. Yet, a careful analysis shows that while individual messages sent by a party might “depend” on all parties in the protocol, each bit sent by a party still depends on only a constant number of parties.

The real issue is that in the protocol of [\[GS18, GIS18\]](#), parties send garbled circuits of circuits whose descriptions depend on all parties. Thus, the resulting garbled circuit may depend on all parties. However, we observe that the way these circuits depend on all parties is very specific. The circuits garbled are keyed circuits of the form $C[v]$, where v is some hardcoded value. C itself is public and does not depend on any party. And while v depends on all parties, each bit of v only depends on a *constant* number of parties! To obtain an input-local MPC protocol, we construct a garbling scheme that has the property that garbling circuits of the form $C[v]$ described above results in a garbled circuit where each bit of the garbled circuit only depends on a constant number of parties. We call such a garbling scheme an input-local garbling scheme. By instantiating the protocol of [\[GS18, GIS18\]](#) with this input-local garbling scheme, we are able to arrive at an input-local MPC protocol.

Combiner-Friendly Homomorphic Secret Sharing (CFHSS). In the sketch of our plan for constructing an FE combiner provided above, we wanted to generate function keys for various circuits with respect to nested FE candidates. As an intermediate tool, we introduce the notion of a combiner-friendly homomorphic secret sharing (CFHSS) scheme. Such an abstraction almost immediately gives rise to an FE combiner, but will be useful in simplifying the presentation of the construction.

Informally, a CFHSS scheme consists of input encoding and function encoding algorithms. The input encoding algorithm runs on an input x and outputs input shares $s_{i,j,k}$ for $i, j, k \in [n]$ (we define CFHSS schemes for triples of indices, since we will require 3-nestings of FE candidates in our construction). The function encoding algorithm runs on a circuit C and outputs function shares $C_{i,j,k}$ for $i, j, k \in [n]$. Then, the decoding algorithm takes as input the evaluation of all shares $C_{i,j,k}(s_{i,j,k})$ and recovers $C(x)$. Informally, the security notion of a CFHSS scheme says that if the shares corresponding to some index i^* remain hidden, then the input is hidden to a computationally bounded adversary and only the evaluation $C(x)$ is revealed.

In order to build an FE combiner from a CFHSS scheme, we will encrypt the share $s_{i,j,k}$ using the nested FE candidate corresponding to indices i, j, k . To provide a function key for a circuit C , we will issue function keys for the circuit $C_{i,j,k}$ with respect to the nested candidate corresponding to indices i, j, k . This allows the decryptor to compute $C_{i,j,k}(s_{i,j,k})$ for all $i, j, k \in [n]$, which by the properties of our CFHSS scheme, is sufficient to determine $C(x)$. Note that in order to argue security, we will have to rely on the Trojan method [ABS15].

Organization. We begin by defining functional encryption, secure multi-party computation, and garbling schemes in Section 2. Then, in Section 3, we define the notion of a functional encryption combiner. In Section 4, we define the notion of an input-local MPC protocol and then show how to construct such a protocol. This is done by constructing a specific garbling scheme that, when used to instantiate the garbling scheme used in the protocol of [GS18, GIS18], results in an input-local MPC protocol. In Section 5, we introduce and define the notion of a combiner-friendly homomorphic secret sharing (CFHSS) scheme and construct such a scheme using an input-local MPC protocol. In Section 6, we construct an FE combiner from a CFHSS scheme. Finally, in Section 7, we observe that our unconditional FE combiner implies a universal FE scheme.

2 Preliminaries

We denote the security parameter by λ . For an integer $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \dots, n\}$. We use $\mathcal{D}_0 \cong_c \mathcal{D}_1$ to denote that two distributions $\mathcal{D}_0, \mathcal{D}_1$ are computationally indistinguishable. We use $\text{negl}(\lambda)$ to denote a function that is negligible in λ . We use $y \leftarrow \mathcal{A}$ to denote that y is the output of a randomized algorithm \mathcal{A} , where the randomness of \mathcal{A} is sampled from the uniform distribution. We write $\mathcal{A}(x; r)$ to denote the output of \mathcal{A} when ran on input x with randomness r . We use PPT as an abbreviation for probabilistic polynomial time.

2.1 Functional Encryption

We define the notion of a (secret key) functional encryption candidate and a (secret key) functional encryption scheme. A functional encryption candidate is associated with the correctness requirement, while a secure functional encryption scheme is associated with both correctness and security.

Syntax of a Functional Encryption Candidate/Scheme. A functional encryption (FE) candidate/scheme FE for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four polynomial time algorithms (Setup, Enc, KeyGen, Dec) defined as follows. Let \mathcal{X}_λ be the input space of the circuit class \mathcal{C}_λ and let \mathcal{Y}_λ be the output space of \mathcal{C}_λ . We refer to \mathcal{X}_λ and \mathcal{Y}_λ as the input and output space of the candidate/scheme, respectively.

- **Setup**, $\text{MSK} \leftarrow \text{FE.Setup}(1^\lambda)$: It takes as input the security parameter λ and outputs the master secret key MSK.
- **Encryption**, $\text{CT} \leftarrow \text{FE.Enc}(\text{MSK}, m)$: It takes as input the master secret key MSK and a message $m \in \mathcal{X}_\lambda$ and outputs CT, an encryption of m .

- **Key Generation**, $\text{SK}_C \leftarrow \text{FE.KeyGen}(\text{MSK}, C)$: It takes as input the master secret key MSK and a circuit $C \in \mathcal{C}_\lambda$ and outputs a function key SK_C .
- **Decryption**, $y \leftarrow \text{FE.Dec}(\text{SK}_C, \text{CT})$: It takes as input a function secret key SK_C , a ciphertext CT and outputs a value $y \in \mathcal{Y}_\lambda$.

Throughout this work, we will only be concerned with *uniform* algorithms. That is, $(\text{Setup}, \text{Enc}, \text{KeyGen}, \text{Dec})$ can be represented as Turing machines (or equivalently uniform circuits).

We describe the properties associated with the above candidate.

Correctness.

Definition 1 (Correctness). *A functional encryption candidate $\text{FE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ is said to be correct if it satisfies the following property: for every $C : \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda \in \mathcal{C}_\lambda, m \in \mathcal{X}_\lambda$ it holds that:*

$$\Pr \left[\begin{array}{l} \text{MSK} \leftarrow \text{FE.Setup}(1^\lambda) \\ \text{CT} \leftarrow \text{FE.Enc}(\text{MSK}, m) \\ \text{SK}_C \leftarrow \text{FE.KeyGen}(\text{MSK}, C) \\ C(m) \leftarrow \text{FE.Dec}(\text{SK}_C, \text{CT}) \end{array} \right] \geq 1 - \text{negl}(\lambda),$$

where the probability is taken over the coins of the algorithms.

IND-Security. We recall indistinguishability-based selective security for FE. This security notion is modeled as a game between a challenger Chal and an adversary \mathcal{A} where the adversary can request functional keys and ciphertexts from Chal . Specifically, \mathcal{A} can submit function queries C and Chal responds with the corresponding functional keys. \mathcal{A} can also submit message queries of the form (x_0, x_1) and receives an encryption of messages x_b for some bit $b \in \{0, 1\}$. The adversary \mathcal{A} wins the game if she can guess b with probability significantly more than $1/2$ and if for all function queries C and message queries (x_0, x_1) , $C(x_0) = C(x_1)$. That is to say, any function evaluation that is computable by \mathcal{A} gives the same value regardless of b . It is required that the adversary must declare the challenge messages at the beginning of the game.

Definition 2 (IND-secure FE). *A secret-key FE scheme FE for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ and message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ is selectively secure if for any PPT adversary \mathcal{A} , there exists a negligible function $\mu(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, the advantage of \mathcal{A} is*

$$\text{Adv}_{\mathcal{A}}^{\text{FE}} = \left| \Pr[\text{Expt}_{\mathcal{A}}^{\text{FE}}(1^\lambda, 0) = 1] - \Pr[\text{Expt}_{\mathcal{A}}^{\text{FE}}(1^\lambda, 1) = 1] \right| \leq \mu(\lambda),$$

where for each $b \in \{0, 1\}$ and $\lambda \in \mathbb{N}$, the experiment $\text{Expt}_{\mathcal{A}}^{\text{FE}}(1^\lambda, b)$ is defined below:

1. **Challenge message queries:** \mathcal{A} submits message queries,

$$\left\{ (x_0^i, x_1^i) \right\}$$

with $x_0^i, x_1^i \in \mathcal{X}_\lambda$ to the challenger Chal .

2. Chal computes $\text{MSK} \leftarrow \text{FE.Setup}(1^\lambda)$ and then computes $\text{CT}_i \leftarrow \text{FE.Enc}(\text{MSK}, x_i^i)$ for all i . The challenger Chal then sends $\{\text{CT}_i\}$ to the adversary \mathcal{A} .
3. **Function queries:** The following is repeated an at most polynomial number of times: \mathcal{A} submits a function query $C \in \mathcal{C}_\lambda$ to Chal. The challenger Chal computes $\text{SK}_C \leftarrow \text{FE.KeyGen}(\text{MSK}, C)$ and sends it to \mathcal{A} .
4. If there exists a function query C and challenge message queries (x_0^i, x_1^i) such that $C(x_0^i) \neq C(x_1^i)$, then the output of the experiment is set to \perp . Otherwise, the output of the experiment is set to b' , where b' is the output of \mathcal{A} .

Adaptive Security. The above security notion is referred to as selective security in the literature. One can consider a stronger notion of security, called *adaptive security*, where the adversary can interleave the challenge messages and the function queries in any arbitrary order. Analogous to [Definition 2](#), we can define an adaptively secure FE scheme. In this paper, we only deal with selectively secure FE schemes. However, the security of these schemes can be upgraded to adaptive with no additional cost [[ABSV15](#)].

Collusions. We can parameterize the FE candidate by the number of function secret key queries that the adversary can make in the security experiment. If the adversary can only submit an a priori upper bounded q secret key queries, we say that the scheme is q -key secure. We say that the functional encryption scheme is unbounded-key secure if the adversary can make an unbounded (polynomial) number of function secret key queries. In this work, we will allow the adversary to make an arbitrary polynomial number of function secret key queries.

FE Candidates vs. FE Schemes. As defined above, an FE scheme must satisfy *both* correctness and security, while an FE candidate is simply the set of algorithms. Unless otherwise specified, we will be dealing with FE candidates that satisfy correctness. We will only refer to FE constructions as FE schemes if it is known that the construction satisfies both correctness and security.

2.2 Secure Multi-Party Computation

The syntax and security definitions for secure multi-party computation can be found in [Appendix A.1](#). In this work, we will deal with protocols that follow a certain structure, introduced in [[GS18](#), [GIS18](#)], called conforming protocols. The full syntactic definition of conforming protocols can be found in [Appendix B](#).

2.3 Garbling Schemes

The definition of garbling schemes can be found in [Appendix A.2](#).

2.4 Correlated Randomness Model

In the correlated randomness model, two parties P_i and P_j are given correlated strings $r_{i,j}$ and $r_{j,i}$, respectively. If we set $r_{i,j} = (k_0, k_1)$ for two strings k_0, k_1 and $r_{j,i} = (b, k_b)$ for a random bit b and

the string k_b , then these two parties can now perform a 2-round information-theoretically secure OT, where P_i is the sender and P_j is the receiver. In the first round, the receiver sends $v = b \oplus c$, where c is the receiver's choice bit. Then, the sender responds with $(y_0, y_1) = (m_0 \oplus k_v, m_1 \oplus k_{1 \oplus v})$. The receiver can then determine m_c by computing $y_c \oplus k_b$.

In this work, we will often say that parties generate correlated randomness necessary to perform a certain number of OTs. By this, we simply mean that the parties repeat the above procedure once for each necessary OT (with the appropriate party as sender/receiver) and use the above 2-round information-theoretically secure OT protocol for each necessary OT.

3 FE Combiners: Definition

In this section, we give a formal definition of an FE combiner. Intuitively, an FE combiner FEComb takes n FE candidates, $\text{FE}_1, \dots, \text{FE}_n$ and compiles them into a new FE candidate with the property that FEComb is a secure FE scheme provided that at least one of the n FE candidates is a secure FE scheme.

Syntax of a Functional Encryption Combiner. A functional encryption combiner FEComb for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four polynomial time algorithms (Setup , Enc , KeyGen , Dec) defined as follows. Let \mathcal{X}_λ be the input space of the circuit class \mathcal{C}_λ and let \mathcal{Y}_λ be the output space of \mathcal{C}_λ . We refer to \mathcal{X}_λ and \mathcal{Y}_λ as the input and output space of the combiner, respectively. Furthermore, let $\text{FE}_1, \dots, \text{FE}_n$ denote the descriptions of n FE candidates.

- **Setup**, $\text{FEComb.Setup}(1^\lambda, \{\text{FE}_i\}_{i \in [n]})$: It takes as input the security parameter λ and the descriptions of n FE candidates $\{\text{FE}_i\}_{i \in [n]}$ and outputs the master secret key MSK .
- **Encryption**, $\text{FEComb.Enc}(\text{MSK}, \{\text{FE}_i\}_{i \in [n]}, m)$: It takes as input the master secret key MSK , the descriptions of n FE candidates $\{\text{FE}_i\}_{i \in [n]}$, and a message $m \in \mathcal{X}_\lambda$ and outputs CT , an encryption of m .
- **Key Generation**, $\text{FEComb.Keygen}(\text{MSK}, \{\text{FE}_i\}_{i \in [n]}, C)$: It takes as input the master secret key MSK , the descriptions of n FE candidates $\{\text{FE}_i\}_{i \in [n]}$, and a circuit $C \in \mathcal{C}_\lambda$ and outputs a function key SK_C .
- **Decryption**, $\text{FEComb.Dec}(\{\text{FE}_i\}_{i \in [n]}, \text{SK}_C, \text{CT})$: It is a deterministic algorithm that takes as input the descriptions of n FE candidates $\{\text{FE}_i\}_{i \in [n]}$, a function secret key SK_C , and a ciphertext CT and outputs a value $y \in \mathcal{Y}_\lambda$.

Remark 1. *In the formal definition above, we have included $\{\text{FE}_i\}_{i \in [n]}$, the descriptions of the FE candidates, as input to all the algorithms of FEComb . For notational simplicity, we will often forgo these inputs and assume that they are implicit.*

We now define the properties associated with an FE combiner. The three properties are correctness, polynomial slowdown, and security. Correctness is analogous to that of an FE candidate, provided that the n input FE candidates are all valid FE candidates. Polynomial slowdown says

that the running times of all the algorithms of FEComb are polynomial in λ and n . Finally, security intuitively says that if at least one of the FE candidates is also secure, then FEComb is a secure FE scheme. We provide the formal definitions below.

Correctness.

Definition 3 (Correctness). *Suppose $\{\text{FE}_i\}_{i \in [n]}$ are correct FE candidates. We say that an FE combiner is correct if for every circuit $C : \mathcal{X}_\lambda \rightarrow \mathcal{Y}_\lambda \in \mathcal{C}_\lambda$, and message $m \in \mathcal{X}_\lambda$ it holds that:*

$$\Pr \left[\begin{array}{l} \text{MSK} \leftarrow \text{FEComb.Setup}(1^\lambda, \{\text{FE}_i\}_{i \in [n]}) \\ \text{CT} \leftarrow \text{FEComb.Enc}(\text{MSK}, \{\text{FE}_i\}_{i \in [n]}, m) \\ \text{SK}_C \leftarrow \text{FEComb.Keygen}(\text{MSK}, \{\text{FE}_i\}_{i \in [n]}, C) \\ C(m) \leftarrow \text{FEComb.Dec}(\{\text{FE}_i\}_{i \in [n]}, \text{SK}_C, \text{CT}) \end{array} \right] \geq 1 - \text{negl}(\lambda),$$

where the probability is taken over the coins of the algorithms and $\text{negl}(\lambda)$ is a negligible function in λ .

Polynomial Slowdown.

Definition 4 (Polynomial Slowdown). *An FE combiner FEComb satisfies polynomial slowdown if on all inputs, the running times of FEComb.Setup, FEComb.Enc, FEComb.Keygen, and FEComb.Dec are at most $\text{poly}(\lambda, n)$, where n is the number of FE candidates that are being combined.*

IND-Security.

Definition 5 (IND-Secure FE Combiner). *An FE combiner FEComb is selectively secure if for any set $\{\text{FE}_i\}_{i \in [n]}$ of correct FE candidates, it satisfies [Definition 2](#), where the descriptions of $\{\text{FE}_i\}_{i \in [n]}$ are public and implicit in all invocations of the algorithms of FEComb, if at least one of the FE candidates $\text{FE}_1, \dots, \text{FE}_n$ also satisfies [Definition 2](#).*

Note that [Definition 2](#) is the IND-security definition for FE.

Robust FE Combiners and Universal FE.

Remark 2. *We also define the notion of a robust FE combiner. An FE combiner FEComb is robust if it is an FE combiner that satisfies the three properties (correctness, polynomial slowdown, and security) associated with an FE combiner when given any set of FE candidates $\{\text{FE}_i\}_{i \in [n]}$, provided that one is a correct and secure FE candidate. No restriction is placed on the other FE candidates. In particular, they need not satisfy correctness at all.*

Robust FE combiners can be used to build a universal functional encryption scheme defined below.

Definition 6 (T -Universal Functional Encryption). *We say that an explicit Turing machine $\Pi_{\text{univ}} = (\Pi_{\text{univ}}.\text{Setup}, \Pi_{\text{univ}}.\text{Enc}, \Pi_{\text{univ}}.\text{KeyGen}, \Pi_{\text{univ}}.\text{Dec})$ is a universal functional encryption scheme parametrized by T if Π_{univ} is a correct and secure FE scheme assuming the existence a correct and secure FE scheme with runtime $< T$.*

4 Input-Local MPC Protocols

As discussed in [Section 1.2](#), if we had a “special” MPC protocol, where every bit of the transcript is computable by a deterministic function on a *constant* number of parties’ inputs and randomness, and the output of the protocol can be computed solely from the transcript, we could use such a protocol to construct an FE combiner. Here, we formally define such a protocol and call it an *input-local* MPC protocol. Since our goal is to construct FE combiners unconditionally, we do not want to assume the existence of OT, so we will define our input-local MPC protocol in the correlated-randomness model.

4.1 Input-Local Protocol Specification

Let Φ be an MPC protocol for n parties P_1, \dots, P_n with inputs x_1, \dots, x_n in the correlated randomness model. We can view Φ as a deterministic MPC protocol, where the input of a party P_i is $(x_i, r_i, (r_{i,j})_{j \neq i})$, where r_i is the randomness used by P_i and $(r_{i,j}, r_{j,i})$ for $i \neq j$ is the correlated randomness tuple used between parties P_i and P_j . Φ is called t -input-local if the following holds:

- **Input-Local Transcript:** Let τ be a transcript of an execution of Φ . Every bit τ_α of τ can be written as a deterministic function of the inputs, randomness, and correlated randomness dependent on at most t parties. That is, there exists a deterministic function f_α such that

$$\tau_\alpha = f_\alpha((x_i)_{i \in \mathcal{S}_\alpha}, (r_i)_{i \in \mathcal{S}_\alpha}, (r_{i,j})_{i,j \in \mathcal{S}_\alpha}),$$

where $|\mathcal{S}_\alpha| \leq t$. If $i \in \mathcal{S}_\alpha$, then τ_α depends on party P_i .

- **Publicly Recoverable Output:** Given a transcript τ of an execution of Φ , there exists a function Eval such that the output of the protocol Φ for all parties is given by

$$y = \text{Eval}(\tau).$$

- **Security:** Φ is simulation secure against $n-1$ semi-honest corruptions, assuming the existence of one-way functions.

No MPC protocol in the literature for all polynomial-sized circuits in the correlated randomness model satisfies the specification of a t -input-local MPC protocol for a constant t . However, the protocols of [\[GS18, GIS18\]](#) come “close”, and we show that with a simple transformation, the protocol of [\[GS18, GIS18\]](#) can be made t -input-local.

[\[GS18, GIS18\]](#) show the following.

Theorem 2 ([\[GS18\]](#),[\[GIS18\]](#),[\[GIS19\]](#)). *Assuming one-way functions, for any circuit C , there exists a 2-round MPC protocol in the correlated randomness model that is secure against semi-honest adversaries that can corrupt up to $n-1$ parties.*

The MPC protocol satisfying [Theorem 2](#) is the MPC protocol of [\[GS18\]](#) modified to operate in the correlated randomness model. In [\[GIS18\]](#), they additionally modify the protocol of [\[GS18\]](#) in other ways, since the focus of [\[GIS18\]](#) is on achieving information-theoretic security for smaller circuit classes and better efficiency. However, one can simply modify the protocol of [\[GS18\]](#) to

operate in the correlated randomness model without making the additional modifications present in [GIS18], a fact which we confirmed with the authors [GIS19].

For completeness, we include a description of the [GS18] protocol in the correlated randomness model in Appendix B.

The MPC protocol of Theorem 2 is not input-local, but can be made input-local via a simple modification. At a high level, the reason that the above protocol is not input-local is because parties P_i , as part of the protocol, send garbled circuits of circuits $C[v]$ that have values v hardcoded in them that depend on $(r_{i,j})_{j \neq i}$, the correlated randomness between P_i and all other parties. As a result, these garbled circuits depend on all parties, and thus, the protocol is not input-local for a constant t . Fortunately, this issue is easily fixable by instantiating the garbling scheme used by the protocol in a specific manner. We consider the garbling scheme for keyed circuits that garbles $C[v]$ by applying Yao's garbling scheme to the universal circuit U , where $U(C, v, x) = C[v](x)$. The garbled circuit of this new scheme consists of \hat{U} , the Yao garbling of U , along with input labels corresponding to C and v . The input labels of this new scheme are the input labels corresponding to x . Observe now that \hat{U} and the input labels for C are clearly input-local, since they only depend on the party P_i that is garbling. Furthermore, since every bit of v only depends on a constant number of parties, each input label for each bit of v also depends on a constant number of parties, giving us an input-local protocol.

Formally, consider the following garbling scheme.

Definition 7 (Input-Local Garbling Scheme). *Let $\text{GC} = (\text{GrbC}, \text{EvalGC})$ denote the standard Yao garbling scheme [Yao86] for poly-sized circuits. Let \mathcal{C} be a class of keyed circuits with key space \mathcal{V} . Let the description length of any $C \in \mathcal{C}$ be ℓ_1 and of any $v \in \mathcal{V}$ be ℓ_2 . Let the input length of any circuit $C \in \mathcal{C}$ be ℓ_3 . Let $\ell = \ell_1 + \ell_2 + \ell_3$. Let C_i, v_i denote the i th bit of the description of C, v , respectively. Let $\text{GC}' = (\text{GrbC}', \text{EvalGC}')$ be a garbling scheme for the class of keyed circuits \mathcal{C} defined as follows:*

- **Garbled Circuit Generation, $\text{GrbC}'(1^\lambda, C[v])$:** *Let U be the universal circuit that, on input (C, v, x) with $|C| = \ell_1, |v| = \ell_2$, and $|x| = \ell_3$, computes $C[v](x)$. Compute $(\hat{U}, (\mathbf{k}_1, \dots, \mathbf{k}_\ell)) \leftarrow \text{GrbC}(1^\lambda, U)$. Output*

$$((\hat{U}, k_1^{C_1}, \dots, k_{\ell_1}^{C_{\ell_1}}, k_{\ell_1+1}^{v_1}, \dots, k_{\ell_1+\ell_2}^{v_{\ell_2}}), (\mathbf{k}_{\ell_1+\ell_2+1}, \dots, \mathbf{k}_\ell)).$$

- **Evaluation, $\text{EvalGC}'(\widehat{C[v]}, (k_1^{x_1}, \dots, k_{\ell_3}^{x_{\ell_3}}))$:** *Parse $\widehat{C[v]}$ as $(\hat{U}, (k_1, k_2, \dots, k_{\ell_1+\ell_2}))$. Run*

$$\text{EvalGC}(\hat{U}, (k_1, \dots, k_{\ell_1+\ell_2}, k_1^{x_1}, \dots, k_{\ell_3}^{x_{\ell_3}}))$$

and output the result.

Correctness of the above garbling scheme follows immediately from the correctness of Yao's garbling scheme and the definition of U . In particular, for every keyed circuit $C[v]$, for any $x \in \{0, 1\}^{\ell_3}$, EvalGC' runs EvalGC on \hat{U} with input labels corresponding to (C, v, x) , giving $U(C, v, x) = C[v](x)$ as desired.

Theorem 3. *The garbling scheme of Definition 7 is secure.*

Proof. Let SimGC be the simulator for Yao’s garbling scheme. The simulator SimGC' operates as follows. Run

$$(\hat{U}, (k_1, \dots, k_\ell)) \leftarrow \text{SimGC}(1^\lambda, \phi(U), C[v](x))$$

and output

$$((\hat{U}, k_1, \dots, k_{\ell_1+\ell_2}), (k_{\ell_1+\ell_2+1}, \dots, k_\ell)).$$

Suppose there exists an adversary \mathcal{A} that can distinguish the output of SimGC' from the real execution. Then, consider the adversary \mathcal{A}' that breaks the security of Yao’s garbling scheme by simply querying its challenger on the pair $(U, (C, v, x))$, rearranging the components of its received challenge to match the output of SimGC' , and running \mathcal{A} . \mathcal{A}' outputs the result of \mathcal{A} . \mathcal{A}' simulates the role of \mathcal{A} ’s challenger exactly and, therefore, must win with nonnegligible advantage, a contradiction. \square

Armed with the above garbling scheme, we are able to obtain an input-local MPC protocol. By taking the MPC protocol of [Theorem 2](#) and instantiating the underlying garbling scheme with the one from [Definition 7](#), we arrive at the following result.

Theorem 4. *Assuming one-way functions, there exists a 3-input-local MPC protocol for any poly-sized circuit C .*

Proof. The proof is deferred to [Appendix C](#).

5 Combiner-Friendly Homomorphic Secret Sharing Schemes

As an intermediate step in our construction of an FE combiner, we define and construct what we call a combiner-friendly homomorphic secret sharing scheme (CFHSS). Informally, a CFHSS scheme consists of input encoding and function encoding algorithms. The input encoding algorithm runs on an input x and outputs input shares $s_{i,j,k}$ for $i, j, k \in [n]$. The function encoding algorithm runs on a circuit C and outputs function shares $C_{i,j,k}$ for $i, j, k \in [n]$. Then, the decoding algorithm takes as input the evaluation of all shares $C_{i,j,k}(s_{i,j,k})$ and recovers $C(x)$. Looking ahead, our CFHSS scheme has several properties that will be useful in constructing an FE combiner. Recall that the high-level idea of our construction was to view each FE candidate as a party P_i in an MPC protocol. In our construction of a CFHSS scheme, each input and function share depends on only the state of a constant number of parties. In particular, share $s_{i,j,k}$ will depend only on the state of parties P_i, P_j , and P_k . Informally, the security notion of a CFHSS scheme says that if the shares corresponding to some index i^* remain hidden, then the input is hidden to a computationally bounded adversary and only the evaluation $C(x)$ is revealed.

5.1 Definition

Definition 8. *A combiner-friendly homomorphic secret sharing scheme, $\text{CFHSS} = (\text{InpEncode}, \text{FuncEncode}, \text{Decode})$, for a class of circuits $\mathcal{C} = \{C_\lambda\}_{\lambda \in \mathbb{N}}$ with input space \mathcal{X}_λ and output space \mathcal{Y}_λ supporting $n \in \mathbb{N}$ candidates consists of the following polynomial time algorithms:*

- **Input Encoding**, $\text{InpEncode}(1^\lambda, 1^n, x)$: *It takes as input the security parameter λ , the number of candidates n , and an input $x \in \mathcal{X}_\lambda$ and outputs a set of input shares $\{s_{i,j,k}\}_{i,j,k \in [n]}$.*

- **Function Encoding**, $\text{FuncEncode}(1^\lambda, 1^n, C)$: It is an algorithm that takes as input the security parameter λ , the number of candidates n , and a circuit $C \in \mathcal{C}$ and outputs a set of function shares $\{C_{i,j,k}\}_{i,j,k \in [n]}$.
- **Decoding**, $\text{Decode}(\{C_{i,j,k}(s_{i,j,k})\}_{i,j,k \in [n]})$: It takes as input a set of evaluations of function shares on their respective input shares and outputs a value $y \in \mathcal{Y}_\lambda \cup \{\perp\}$.

A combiner-friendly homomorphic secret sharing scheme, CFHSS, is required to satisfy the following properties:

- **Correctness**: For every $\lambda \in \mathbb{N}$, circuit $C \in \mathcal{C}_\lambda$, and input $x \in \mathcal{X}_\lambda$, it holds that:

$$\Pr \left[\begin{array}{l} \{s_{i,j,k}\}_{i,j,k \in [n]} \leftarrow \text{InpEncode}(1^\lambda, 1^n, x) \\ \{C_{i,j,k}\}_{i,j,k \in [n]} \leftarrow \text{FuncEncode}(1^\lambda, 1^n, C) \\ C(x) \leftarrow \text{Decode}(\{C_{i,j,k}(s_{i,j,k})\}_{i,j,k \in [n]}) \end{array} \right] \geq 1 - \text{negl}(\lambda),$$

where the probability is taken over the coins of the algorithms and $\text{negl}(\lambda)$ is a negligible function in λ .

- **Security**:

Definition 9 (IND-secure CFHSS). A combiner-friendly homomorphic secret sharing scheme CFHSS for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ and input space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ is selectively secure if for any PPT adversary \mathcal{A} , there exists a negligible function $\mu(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, the advantage of \mathcal{A} is

$$\text{Adv}_{\mathcal{A}}^{\text{CFHSS}} = \left| \Pr[\text{Expt}_{\mathcal{A}}^{\text{CFHSS}}(1^\lambda, 1^n, 0) = 1] - \Pr[\text{Expt}_{\mathcal{A}}^{\text{CFHSS}}(1^\lambda, 1^n, 1) = 1] \right| \leq \mu(\lambda),$$

where for each $b \in \{0, 1\}$ and $\lambda \in \mathbb{N}$ and $n \in \mathbb{N}$, the experiment $\text{Expt}_{\mathcal{A}}^{\text{CFHSS}}(1^\lambda, 1^n, b)$ is defined below:

1. **Secure share**: \mathcal{A} submits an index $i^* \in [n]$ that it will not learn the input shares for.
2. **Challenge input queries**: \mathcal{A} submits input queries,

$$\left(x_0^\ell, x_1^\ell \right)_{\ell \in [L]}$$

with $x_0^\ell, x_1^\ell \in \mathcal{X}_\lambda$ to the challenger Chal , where $L = \text{poly}(\lambda)$ is chosen by \mathcal{A} .

3. For all ℓ , Chal computes $\{s_{i,j,k}^\ell\}_{i,j,k \in [n]} \leftarrow \text{InpEncode}(1^\lambda, 1^n, x_0^\ell)$. For all ℓ , the challenger Chal then sends $\{s_{i,j,k}^\ell\}_{i,j,k \in [n] \setminus \{i^*\}}$, the input shares that do not correspond to i^* , to the adversary \mathcal{A} .
4. **Function queries**: The following is repeated an at most polynomial number of times: \mathcal{A} submits a function query $C \in \mathcal{C}_\lambda$ to Chal . The challenger Chal computes function shares $\{C_{i,j,k}\}_{i,j,k \in [n]} \leftarrow \text{FuncEncode}(1^\lambda, 1^n, C)$ and sends them to \mathcal{A} along with all evaluations $\{C_{i,j,k}(s_{i,j,k}^\ell)\}_{i,j,k \in [n]}$ for all $\ell \in [L]$.
5. If there exists a function query C and challenge message queries (x_0^ℓ, x_1^ℓ) such that $C(x_0^\ell) \neq C(x_1^\ell)$, then the output of the experiment is set to \perp . Otherwise, the output of the experiment is set to b' , where b' is the output of \mathcal{A} .

5.2 Construction

Using 3-input-local MPC protocols $\{\Phi_C\}$ for a circuit class \mathcal{C} and a PRF, we will construct a combiner-friendly homomorphic secret sharing scheme for \mathcal{C} . For a circuit $C \in \mathcal{C}$ and number of parties n , we say that Φ_C is an MPC protocol for C on n parties if it computes the function $C(x_1 \oplus \dots \oplus x_n)$ on inputs x_1, \dots, x_n .

Formally, we show the following.

Theorem 5. *Given 3-input-local MPC protocols $\{\Phi_C\}$ for a circuit class \mathcal{C} and assuming one-way functions, there exists a combiner-friendly homomorphic secret sharing scheme for \mathcal{C} for $n = \text{poly}(\lambda)$ candidates.*

Using [Theorem 4](#) to instantiate the 3-input-local MPC protocols, we immediately arrive at the following.

Theorem 6. *Assuming one-way functions, there exists a combiner-friendly homomorphic secret sharing scheme for \mathcal{P}/poly for $n = \text{poly}(\lambda)$ candidates.*

Notation:

- Let PRF be a pseudorandom function with λ -bit keys that takes λ -bit inputs and outputs in $\{0, 1\}^*$. PRF will be used to generate the randomness needed for various randomized algorithms. As the length of randomness needed varies by use case (but is always polynomial in length), we don't specify the output length of PRF here and the output length needed will be clear from context. It is easy to build our required pseudorandom function from one with a fixed length output. Let PRF' be a pseudorandom function that maps $\{0, 1\}^{2\lambda}$ -bit inputs to a single output bit in $\{0, 1\}$. Then, to evaluate PRF(K, x) to an appropriate output length ℓ , we would simply compute the output bit by bit by evaluating PRF'($K, x||1$), PRF'($K, x||2$), \dots , PRF'($K, x||\ell$). When we write $(r_1, r_2, r_3) := \text{PRF}(K, x)$, we mean that we generate the randomness needed for three different algorithms using this PRF, where the length of each r_i depends on the amount of randomness needed by the algorithm. This can be done in the same manner, by computing r_i bit by bit by evaluating PRF'($K, x||i||1$), PRF'($K, x||i||2$), \dots etc.
- For a 3-input-local protocol Φ for a circuit $C \in \mathcal{C}$, we use the same syntax as in [Section 4](#) to refer to the various components and algorithms associated with this protocol. We implicitly assume that the description of the 3-input-local protocol Φ for C is included in the descriptions of the function shares for C .
- Let $\text{Corr}(1^\lambda, 1^\ell, i, j) \rightarrow (r_{i,j}, r_{j,i})$ be the function that on input the security parameter λ , a length parameter ℓ , and indices $i \neq j \in [n]$ outputs correlated random strings $r_{i,j}$ and $r_{j,i}$ each in $\{0, 1\}^\ell$. We will assume that $i < j$ and if not, we implicitly assume that the indices are swapped when evaluating the algorithm. Looking ahead, ℓ is set as the the length of the correlated randomness required between two parties in the execution of the 3-input-local protocol. For simplicity, we will omit the parameter ℓ in the description below when it is clear from the context. We note that Corr can be implemented by generating random OT-correlations.

- In the construction, for simplicity, we will denote input and function shares for the tuple of indices (i, i, i) by s_i and C_i , respectively. Similarly, we will denote the input and function shares for the tuple of indices (i, j, i) with $i \neq j$ by $s_{i,j}$ and $C_{i,j}$, respectively. We will denote input and function shares for the tuple of indices (i, j, k) with $i \neq j \neq k$ by $s_{i,j,k}$ and $C_{i,j,k}$ respectively. All other input and function shares are set to \perp .

Overview: We provided a sketch of our construction in [Section 1.2](#). Here, we provide more details to assist in the understanding of our construction. The input encoding algorithm will take an input x , n -out-of- n secret share it into shares x_1, \dots, x_n , sample PRF keys K_i for $i \in [n]$ and shared PRF keys K_{ij} for $i < j \in [n]$. Shares of the form s_i will be (x_i, K_i) , shares of the form $s_{i,j}$ will be $(x_i, x_j, K_i, K_j, K_{ij})$, and shares of the form $s_{i,j,k}$ will be $(x_i, x_j, x_k, K_i, K_j, K_k, K_{ij}, K_{ik}, K_{jk})$. These will serve as the inputs to the function shares $\{C_{i,j,k}\}_{i,j,k \in [n]}$. Intuitively, a share $s_{i,j,k}$ (or $s_{i,j}$ or s_i) contains all the input shares and PRF keys that correspond to the indices i, j, k (or i, j or i).

The description of function shares of the form C_i , $C_{i,j}$, and $C_{i,j,k}$ is given in [Figure 3](#), [Figure 4](#), and [Figure 5](#), respectively. The purpose of C_i , $C_{i,j}$, and $C_{i,j,k}$ is to simply output input-local bits in the transcript of Φ_C dependent on either only P_i , the two parties P_i and P_j , or the three parties P_i, P_j, P_k , respectively.

Given evaluations of all the function shares, decoding operates by using the evaluations to obtain a transcript τ of an execution of Φ_C and then running the evaluation procedure of Φ_C .

Construction: We now provide the formal construction.

- **Input Encoding, $\text{InpEncode}(1^\lambda, 1^n, x)$:**
 - XOR secret share x uniformly at random across n shares such that $x_1 \oplus \dots \oplus x_n = x$.
 - For $i \leq j \in [n]$, sample distinct PRF keys K_{ij} . For $i > j \in [n]$, set $K_{ij} = K_{ji}$. Set $K_i = K_{ii}$.
 - For $i \in [n]$, set $s_i = (x_i, K_i)$.
 - For $i, j \in [n]$ with $i < j$, set $s_{i,j} = (x_i, x_j, K_i, K_j, K_{ij})$.
 - For $i, j, k \in [n]$ with $i < j < k$, set $s_{i,j,k} = (x_i, x_j, x_k, K_i, K_j, K_k, K_{ij}, K_{ik}, K_{jk})$.
 - Set all other shares to \perp .
 - Output all shares $\{s_{i,j,k}\}_{i,j,k \in [n]}$.
- **Function Encoding, $\text{FuncEncode}(1^\lambda, 1^n, C)$:** Let Φ denote the 3-input-local MPC protocol for C on n parties. For every bit τ_α in τ , a transcript of Φ , let \mathcal{S}_α denote the set of parties that τ_α depends on and f_α be the function that computes τ_α with respect to these parties' inputs and randomness (see [Section 4](#)).
 - Sample tag tag_{rand} from $\{0, 1\}^\lambda$, uniformly at random.
 - For $i \in [n]$, function share C_i is given by circuit C_i in [Figure 3](#).
 - For $i, j \in [n]$ with $i < j$, function share $C_{i,j}$ is given by circuit $C_{i,j}$ in [Figure 4](#).
 - For $i, j, k \in [n]$ with $i < j < k$, function share $C_{i,j,k}$ is given by circuit $C_{i,j,k}$ in [Figure 5](#).

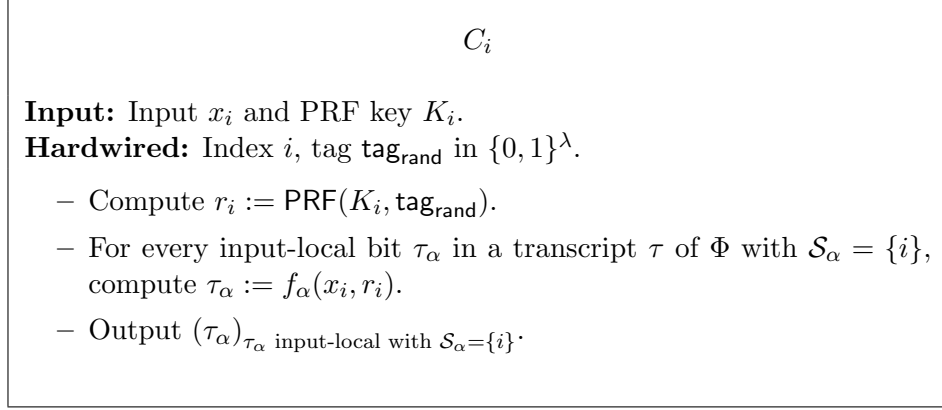


Figure 3: Description of Function Share C_i .

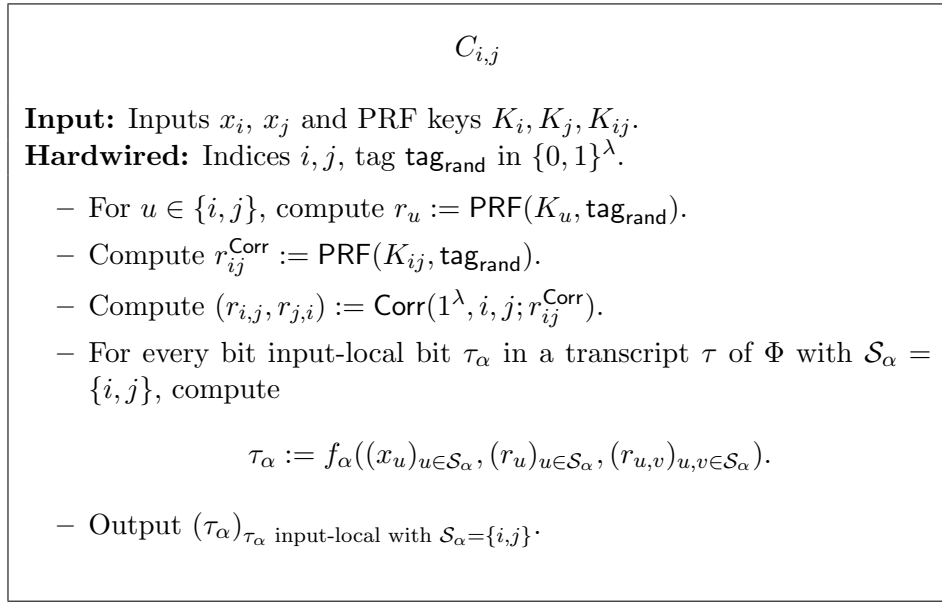


Figure 4: Description of Function Share $C_{i,j}$.

– Set all other function shares to \perp and output $\{C_{i,j,k}\}_{i,j,k \in [n]}$.

• **Decoding**, $\text{Decode}(\{C_{i,j,k}(s_{i,j,k})\}_{i,j,k \in [n]})$: It does the following:

- Rearrange all input-local bits τ_α output by the function shares to obtain τ , the transcript of an execution of Φ .
- Run $\text{Eval}(\tau)$ to obtain the output y .

Correctness: Correctness follows from the correctness of the underlying set of 3-input-local MPC protocols $\{\phi_C\}$. In particular, for any circuit $C \in \mathcal{C}_\lambda$ and input $x \in \mathcal{X}_\lambda$, we note that the Decode algorithm obtains τ , the transcript of an execution of ϕ_C . Therefore, by running Eval on τ , Decode

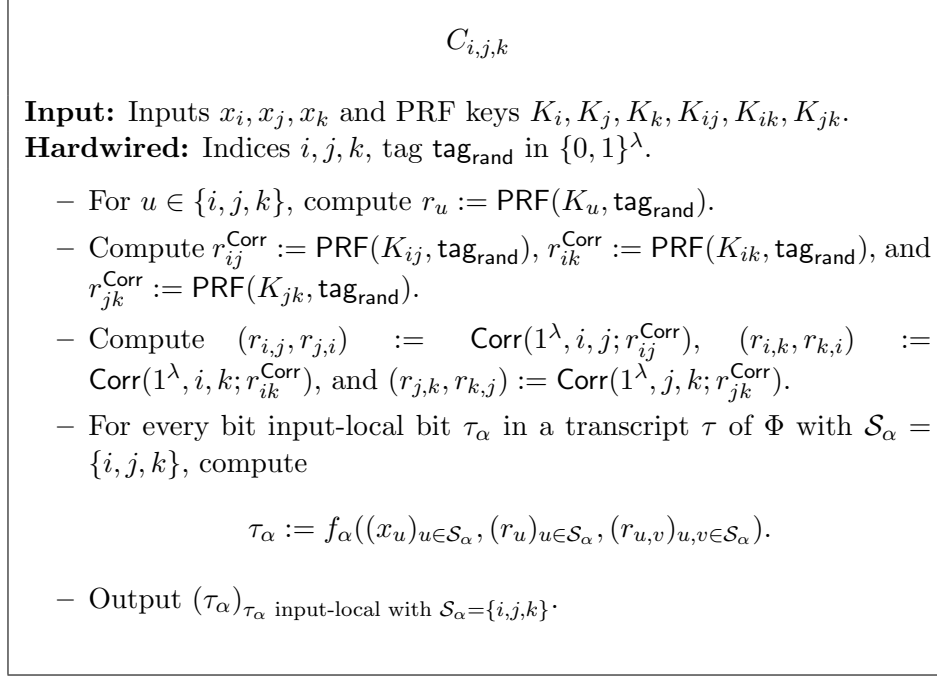


Figure 5: Description of Circuit $C_{i,j,k}$.

obtains

$$y = C(x_1 \oplus \dots \oplus x_n) = C(x)$$

as desired.

Security: We will argue the security of our CFHSS scheme via a series of hybrid games. Throughout, we use i^* to denote the index of the secure share. For simplicity, we only consider the case where the adversary submits only one input and one function query, but the proof immediately extends to a polynomial number of input and function queries. Consider the following sequence of hybrid games.

Hyb₀: This hybrid corresponds to the real security game where the challenger sets the bit b to 0.

Hyb₁: This hybrid is the same as **Hyb₀** except that whenever the challenger would need to generate randomness by running $\text{PRF}(K, \cdot)$ for $K = K_{i^*}, K_{i^*1}, \dots, K_{i^*n}$, where i^* is secure share index chosen by the adversary, to compute a function share evaluation, the challenger instead samples the randomness uniformly. This can be viewed as replacing each function $\text{PRF}(K, \cdot)$ for the keys K listed above by a truly random function f_K .

Hyb₂: In this hybrid, we change how the outputs of function share evaluations dependent on i^* ($C_{i,j,k}(s_{i,j,k})$ for one of $i, j, k = i^*$, $C_{i,j}(s_{i,j})$ for one of $i, j = i^*$, and $C_{i^*}(s_{i^*})$) are computed. The challenger now runs the protocol Φ_C on inputs $(x_i, r_i, (r_{i,j})_{j \neq i})$. For r_i with $i \neq i^*$, $r_i = \text{PRF}(K_i, \text{tag}_{\text{rand}})$ (as done in the function shares dependent on i). For $r_{i,j}$ with both $i, j \neq i^*$, $(r_{i,j}, r_{j,i})$ is computed by $\text{Corr}(1^\lambda, i, j, \text{PRF}(K_{ij}, \text{tag}_{\text{rand}}))$. r_{i^*} is generated uniformly at random and

$(r_{i^*,j}, r_{j,i^*})$ is computed by running $\text{Corr}(1^\lambda, i^*, j)$ with uniform randomness. From this execution of Φ_C , the challenger obtains τ . It uses the bit values in τ to determine the output values for the bits dependent on i^* .

Hyb₃: In this hybrid, the challenger obtains the transcript τ by running

$$\text{Sim}_{\Phi_C}(C(x), \{(x_i, r_i, (r_{i,j})_{j \neq i^*})\}_{i \neq i^*}),$$

with the randomness set as before via the appropriate PRF evaluations.

Lemma 1. *If PRF is a λ -bit pseudorandom function, then Hyb₀ and Hyb₁ are computationally indistinguishable.*

Proof. This can be shown by a series of n consecutive sub-hybrids $\text{Hyb}_{0,i}$, where in each sub-hybrid, we switch replace every evaluation of $\text{PRF}(K, \cdot)$ for K one of $K_{j^*,1}, \dots, K_{i^*,n}$ (note $K_{i^*} = K_{i^*,i^*}$) with a uniformly random function f_K . Suppose there exists an adversary \mathcal{A} that can distinguish between any two consecutive sub-hybrids. Then, we construct an adversary \mathcal{A}' that can break the pseudorandomness property of PRF. \mathcal{A}' simply runs \mathcal{A} and plays the role of the challenger. Whenever \mathcal{A}' would run $\text{PRF}(K, \cdot)$ on some input tag , \mathcal{A}' instead queries its oracle on tag and uses this value as its output. If \mathcal{A} thinks it is seeing $\text{Hyb}_{0,i}$, \mathcal{A}' outputs that its oracle is the PRF and if \mathcal{A} thinks it is seeing $\text{Hyb}_{0,i+1}$, \mathcal{A}' outputs that its oracle is a truly random function. Note that \mathcal{A}' simulates these hybrids exactly, so \mathcal{A}' will win with nonnegligible advantage, contradicting the pseudorandomness of PRF. Since $n = \text{poly}(\lambda)$, it follows that $\text{Hyb}_0 = \text{Hyb}_{0,0} \cong_c \text{Hyb}_{0,n} = \text{Hyb}_1$. \square

Lemma 2. *Hyb₁ and Hyb₂ are indistinguishable.*

Proof. Observe that Hyb_1 and Hyb_2 are identically distributed, and therefore are indistinguishable to \mathcal{A} . In Hyb_2 , we simply determined τ by running the MPC protocol Φ_C and the steps of Φ_C to compute τ are identical in Hyb_1 as they are in the execution of Φ_C . \square

Lemma 3. *If Φ_C is a 3-input-local MPC protocol, then Hyb₂ and Hyb₃ are computationally indistinguishable.*

Suppose there exists an adversary \mathcal{A} that can distinguish between these two hybrids. Then, we construct an adversary \mathcal{A}' that can break the security of Φ_C . \mathcal{A}' simply runs \mathcal{A} and plays the role of the challenger in Hyb_2 . \mathcal{A}' queries its challenger and provides it with $(C(x), \{(x_i, r_i, \{r_{i,j}\}_{j \neq i^*})\}_{i \in [n] \setminus \{i^*\}})$ and the auxiliary input x_{i^*} . For all parties P_i for $i \neq i^*$, \mathcal{A}' generates their randomness by $\text{PRF}(K_i, \text{tag}_{\text{rand}})$ and for all pairs of parties P_i, P_j with $i, j \neq i^*$, \mathcal{A}' generates their correlated randomness by $\text{Corr}(1^\lambda, i, j, \text{PRF}(K_{ij}, \text{tag}_{\text{rand}}))$. \mathcal{A}' obtains τ from its challenger. It then uses this information to play the role of the challenger for \mathcal{A} . If \mathcal{A} thinks it is in Hyb_2 then \mathcal{A}' outputs that it is seeing the transcript of the real execution of the protocol and if \mathcal{A} thinks it is in Hyb_3 , then \mathcal{A}' outputs that it is seeing the transcript of a simulated execution of the protocol. Note that \mathcal{A}' simulates these hybrids exactly, so \mathcal{A}' will win with nonnegligible advantage, contradicting the security of Φ_C . \square

Note that Hyb_3 is now independent of x since $\{x_i\}_{i \in [n] \setminus \{i^*\}}$ information-theoretically hides x and Hyb_3 is independent of x_{i^*} . By an analogous series of hybrids with $b = 1$, it follows that the real game with $b = 0$ is computationally indistinguishable from the real game with $b = 1$. \square

Efficiency and Security Against Stronger Adversaries: In this work, we focus on security against polynomial sized adversaries. However, one could consider security against adversaries of size $O(s(\lambda))$, where $s(\lambda)$ is larger than polynomial. In this case, our CFHSS scheme is secure if the underlying function is also secure against adversaries of size $O(s(\lambda))$. More specifically, if no adversary of size $O(s(\lambda))$ can invert the one-way function with probability greater than $\epsilon(\lambda)$, then the advantage of an adversary of size $O(s(\lambda))$ in the CFHSS security game is bounded by $\epsilon(\lambda) \cdot \text{poly}(\lambda)$ for some fixed polynomial. To see this, note that our hybrids only rely on the security of the OWF along with the simulator for Φ_C (which, in turn, only relies on the security of the OWF). The simulator for Φ_C operates by simulating garbled circuits, and security for garbled circuits generalizes in this manner. Thus, the same proofs extend to prove security against $O(s(\lambda))$ size adversaries.

Here, we discuss some efficiency properties associated with our CFHSS scheme. First, observe that the size of the input encoding circuit $\text{InpEncode}(\cdot)$ does not depend on the size of the circuit class. Moreover, the size of any function share scales linearly with the size of the circuit computing a transcript of Φ . If we instantiate the underlying conforming protocol with [GMW87] (which operates gate by gate and, therefore, the size of the transcript scales linearly with the size of the circuit), the 3-input-local MPC protocol Φ_C of Theorem 4 will have a transcript τ of size that depends linearly on $|C|$. Thus, the size of any function share can be bounded by $|C| \cdot \text{poly}(\lambda, n)$ for some fixed polynomial. Formally, we have the following, where we use the notation $(s(\lambda), \epsilon(\lambda))$ -secure to mean that no adversary of size $s(\lambda)$ can have advantage greater than $\epsilon(\lambda)$ in the security game.

Theorem 7. *Assuming an $(O(s(\lambda)), \epsilon(\lambda))$ -secure one-way function for $s(\lambda) = \omega(\text{poly}(\lambda))$, there exists an $(O(s(\lambda)), \text{poly}(\lambda) \cdot \epsilon(\lambda))$ -secure combiner-friendly homomorphic secret sharing scheme for P/poly for $n = O(\text{poly}(\lambda))$ candidates. Moreover, the size of InpEncode is independent of the size of the circuit class and the size of any $C_{i,j,k}$ is bounded by $|C| \cdot \text{poly}(\lambda, n)$ for some fixed polynomial.*

6 Construction of an FE Combiner from a CFHSS Scheme

In this section, we show how to use a CFHSS scheme and one-way functions to construct an FE combiner. Instantiating the CFHSS scheme with the construction in Section 5 and the one-way function with the concatenation of the one-way function candidates implied by our FE candidates (as described in Section 1.2), we arrive at the following result.

Theorem 8. *There exists an unconditionally secure unbounded-key FE combiner for $n = \text{poly}(\lambda)$ FE candidates for P/poly .*

In the rest of this section, we show Theorem 8.

6.1 d-Nested FE

A tool used in our construction is d -nested FE (for $d = 3$). d -nested FE is a new FE candidate that can be created easily from d FE candidates by simply encrypting in sequence using the d FE

candidates. Intuitively, this new FE candidate will be secure as long as one of the d candidates is secure since an adversary should be unable to break the encryption of the secure candidate. d -nested FE can be viewed as an FE combiner that can only handle a constant number of FE candidates since the runtime of its algorithms may depend exponentially on d . We present the definition of d -nested FE below.

Definition 10 (d -nested FE). *Let FE_1, \dots, FE_d be d correct FE candidates. We define another FE candidate FE_S , where $S = [1, \dots, d]$, as follows:*

- $\text{Setup}(1^\lambda)$: *It computes $MSK_i \leftarrow FE_i.\text{Setup}(1^\lambda)$ for all $i \in [n]$ and outputs $MSK = \{MSK_i\}_{i \in [n]}$ as the master secret key for FE_S .*
- $\text{Enc}(MSK = \{MSK_1, \dots, MSK_d\}, m)$: *It first computes $CT_1 = FE_1.\text{Enc}(MSK_1, m)$ and then recursively computes $CT_{i+1} = FE_{i+1}.\text{Enc}(MSK_{i+1}, CT_i)$ for $i \in [d-1]$. It outputs CT_d as the ciphertext.*
- $\text{KeyGen}(MSK = \{MSK_1, \dots, MSK_d\}, C)$: *On input the master secret key MSK and a circuit $C \in \mathcal{C}$, it first computes $SK_1 = FE_1.\text{KeyGen}(MSK_1, C)$ and sets $G_1 = FE_1.\text{Dec}(SK_1, \cdot)$ as a circuit. Then it recursively computes $SK_{i+1} = FE_{i+1}.\text{KeyGen}(MSK_{i+1}, G_i)$ for $i \in [d-1]$ where circuit $G_i = FE_i.\text{Dec}(SK_i, \cdot)$. It outputs SK_d as the function secret key for the circuit C .*
- $\text{Dec}(SK_d, CT_d)$: *It outputs $FE_d.\text{Dec}(SK_d, CT_d)$.*

Theorem 9 ([ABJ⁺19]). *For constant d , FE_S , defined by Definition 10, is an FE candidate. Moreover, if at least one of the d FE candidates is secure, then FE_S is also secure (it is an FE scheme).*

For completeness, we include the proof of Theorem 9 from [ABJ⁺19] in Appendix D.

6.2 Construction

We now formally describe the construction. First, we provide some notation that will be used throughout the construction.

Notation:

- Let FE_1, \dots, FE_n denote n FE candidates. In the following construction, we assume that the descriptions $\{FE_i\}_{i \in [n]}$ are implicit in all the algorithms of FEComb.
- Let FE_{ijk} denote the 3-nested FE candidate derived by nesting FE_i , FE_j , and FE_k .
- Let $\text{CFHSS} = (\text{InpEncode}, \text{FuncEncode}, \text{Decode})$ be a combiner-friendly homomorphic secret sharing scheme. Let ℓ_{output} denote the length of the outputs obtained from the evaluation of function shares on input shares.
- Let E be any λ -bit CPA-secure secret-key encryption scheme with message space $\{0, 1\}^{\ell_{\text{output}}}$.

- Let $\ell_x = \ell_x(\lambda)$ denote the length of the messages and let $\ell_E = \ell_E(\lambda)$ denote the length of the encryption key for the scheme E.

Construction:

- **FECOMB.Setup(1^λ)** : On input the security parameter, it runs $\text{FE}_{ijk}.\text{Setup}(1^\lambda)$ for $i, j, k \in [n]$ and $\text{E.SK} \leftarrow \text{E.Setup}(1^\lambda)$. It outputs $\text{MSK} = (\{\text{MSK}_{ijk}\}_{i,j,k \in [n]}, \text{E.SK})$.
- **FECOMB.Enc($\text{MSK}, x \in \{0, 1\}^{\ell_x}$)** : It executes the following steps.
 - First, encode x into n^3 shares by running $\text{CFHSS.InpEncode}(1^\lambda, 1^n, x)$ to compute $\{s_{i,j,k}\}_{i,j,k \in [n]}$. Then, for all $i, j, k \in [n]$, compute
$$\text{CT}_{ijk} = \text{FE}_{ij}.\text{Enc}\left(\text{MSK}_{ijk}, (s_{i,j,k}, 0^{\ell_E}, 0)\right).$$
 - Output $\text{CT} = \{\text{CT}_{ijk}\}_{i,j,k \in [n]}$.
- **FECOMB.KeyGen(MSK, C)** : It executes the following steps.
 - For all $i, j, k \in [n]$, it computes $c_{i,j,k} \leftarrow \text{E.Enc}(\text{E.SK}, 0^{\ell_{\text{output}}})$, where ℓ_{output} is the length of evaluations of function shares on input shares of CFHSS.
 - It computes $\{C_{i,j,k}\}_{i,j,k \in [n]} \leftarrow \text{CFHSS.FuncEncode}(1^\lambda, 1^n, C)$.
 - For all $i, j, k \in [n]$, it computes $\text{SK}_{H_{i,j,k}} \leftarrow \text{FE}_{ijk}.\text{KeyGen}(\text{MSK}_{ijk}, H_{i,j,k})$, where circuit $H_{i,j,k}$ is described in [Figure 6](#).
 - It outputs $\text{SK}_C = (\{\text{SK}_{H_{i,j,k}}\}_{i,j,k \in [n]})$.

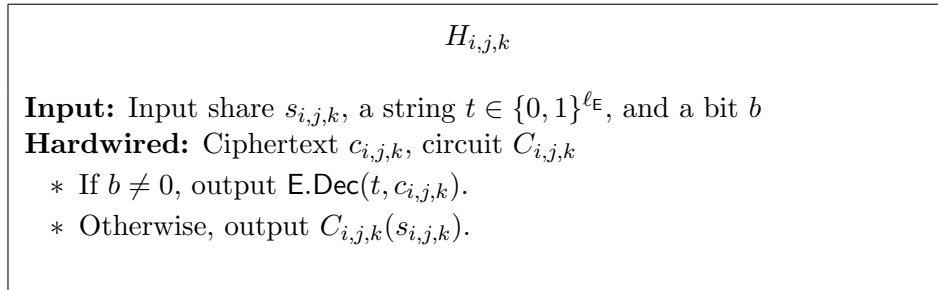


Figure 6: Description of the Evaluation Circuit.

- **FECOMB.Dec(SK_C, CT)** : Parse SK_C as $(\{\text{SK}_{H_{i,j,k}}\}_{i,j,k \in [n]})$ and CT as $\{\text{CT}_{ijk}\}_{i,j,k \in [n]}$. For all $i, j, k \in [n]$, compute $y_{i,j,k} = \text{FE}_{ijk}.\text{Dec}(\text{SK}_{H_{i,j,k}}, \text{CT}_{ijk})$. Run $\text{CFHSS.Decode}(\{y_{i,j,k}\}_{i,j,k \in [n]})$ and output the result.

Correctness: Correctness follows from the correctness of CFHSS and the fact that all correct encryptions are encryptions of messages of the form $(s_{i,j,k}, 0^{\ell_E}, 0)$. In particular, for all $i, j, k \in [n]$, $H_{i,j,k}(s_{i,j,k}, 0^{\ell_E}, 0) = C_{i,j,k}(s_{i,j,k})$ and then $\text{CFHSS.Decode}(\{C_{i,j,k}(s_{i,j,k})\}_{i,j,k \in [n]}) = C(x)$ by the correctness of CFHSS.

Polynomial Slowdown: The fact that all the algorithms of FEComb run in time $\text{poly}(\lambda, n)$ is immediate from the efficiency of the FE candidates, CFHSS, and E and the fact that there are $n^3 = \text{poly}(n)$ different tuples (i, j, k) for $i, j, k \in [n]$.

6.3 Security Proof

For simplicity, we will consider the case where the adversary submits a single message pair. This proof naturally extends to multiple message pairs via standard hybrid techniques (flip each message pair one at a time to argue indistinguishability for a polynomial number of message pairs). We show that any PPT adversary \mathcal{A} will only succeed in the FE selective security game (Definition 2) with negligible probability. We will show this via a sequence of hybrids. Assume that the i^* th FE candidate, FE_{i^*} is secure.

Hyb₀: This hybrid corresponds to the real security game where the challenger sets the bit b to 0.

Hyb₁: This hybrid is the same as **Hyb₀** except that when generating any $c_{i,j,k}$ when generating a function key for a circuit C , the challenger computes $c_{i,j,k} \leftarrow \text{E.Enc}(\text{E.SK}, \beta_{i,j,k})$, where

$$\beta_{i,j,k} = C_{i,j,k}(s_{i,j,k}),$$

the value obtained by running the (i, j, k) th function share of C on the (i, j, k) th input share of x_0 . Note that the challenger knows the $s_{i,j,k}$ values because \mathcal{A} must submit x_0 and the challenger generates the encryption of x_0 prior to generating the function key SK_C .

Hyb₂: This hybrid is the same as **Hyb₁** except that when encrypting x , whenever the challenger generates CT_{ijk} for some pair (i, j, k) with $i = i^*$, $j = i^*$, or $k = i^*$, **Chal** does so by setting

$$\text{CT}_{ijk} = \text{FE}_{ijk}.\text{Enc}(\text{MSK}_{ijk}, (r_{i,j,k}, \text{E.SK}, 1)),$$

where $r_{i,j,k}$ is a uniformly random string of the same length as the input share $s_{i,j,k}$. That is, $s_{i,j,k}$ is replaced with $r_{i,j,k}$, the 0^{ℓ_E} term is replaced with E.SK , the secret key for E , and the last bit is switched to 1 when generating ciphertexts for pairs of indices that contain i^* .

Hyb₃: This hybrid is the same as **Hyb₂** except that the challenger sets the input shares $s_{i,j,k}$ by running $\text{CFHSS.InpEncode}(1^\lambda, 1^n, x_1)$ instead of $\text{CFHSS.InpEncode}(1^\lambda, 1^n, x_0)$.

Hyb₄: This hybrid is the same as **Hyb₃** except that when encrypting x , whenever the challenger generates CT_{ijk} for some pair (i, j, k) with $i = i^*$, $j = i^*$, or $k = i^*$, **Chal** does so by setting

$$\text{CT}_{ijk} = \text{FE}_{ijk}.\text{Enc}\left(\text{MSK}_{ijk}, (s_{i,j,k}, 0^{\ell_E}, 0)\right).$$

Hyb₅: This hybrid is the same as **Hyb₄** except that when generating $c_{i,j,k}$ when generating a function key for a circuit C , the challenger computes $c_{i,j,k} \leftarrow \text{E.Enc}(\text{E.SK}, 0^{\ell_{\text{output}}})$.

Hyb₆: This hybrid corresponds to the real security game where the challenger sets the bit b to 1.

Lemma 4. *If E is a λ -bit CPA-secure secret-key encryption scheme, then Hyb_0 and Hyb_1 are computationally indistinguishable.*

Proof. Suppose there exists an adversary \mathcal{A} that can distinguish between these two hybrids. Then, we construct an adversary \mathcal{A}' that can break the security game of E . Since \mathcal{A} can distinguish between Hyb_0 and Hyb_1 and since there are a polynomial number of pairs (i, j, k) , we can construct a sequence of hybrids that correspond to switching the encrypted values of $c_{i,j,k}$ for each of the function keys one at a time. It follows that \mathcal{A} must be able to distinguish between two of these neighboring hybrids. WLOG, say \mathcal{A} distinguishes between the sequential hybrids where the encrypted value of $c_{i',j',k'}$ value is changed for the α th function key. Then, \mathcal{A}' runs as follows. It runs \mathcal{A} and plays the role of the challenger. Whenever, it would have to generate a ciphertext by running $E.\text{Enc}$, it instead queries its challenger on the message to receive the ciphertext. When it wants to generate $c_{i',j',k'}$, it submits the message pair $(0^{\ell_{\text{output}}}, \beta_{i',j',k'})$ to its challenger where the first component is the unchanged (all zero) value and the second component is the changed evaluation value corresponding to Hyb_0 and Hyb_1 , respectively. It then uses the ciphertext it receives from its challenger. When \mathcal{A} outputs, \mathcal{A}' outputs the same response. Note that \mathcal{A}' simulates \mathcal{A} on these two neighboring hybrids perfectly and so \mathcal{A}' will distinguish with nonnegligible advantage, contradicting the security of E . Therefore, Hyb_0 and Hyb_1 must be indistinguishable. \square

Lemma 5. *If FE_{i^*} is a selectively-secure FE scheme, then Hyb_1 and Hyb_2 are computationally indistinguishable.*

Proof. Suppose there exists an adversary \mathcal{A} that can distinguish between these two hybrids. Then, we construct an adversary \mathcal{A}' that can break the security game of FE_{i^*} . \mathcal{A}' runs \mathcal{A} and simulates the role of the challenger. Whenever, it needs to encrypt using $\text{FE}_{i^*}.\text{Enc}$, it computes the messages m_1 and m_2 that it would want to encrypt were it simulating Hyb_1 or Hyb_2 , respectively. It then submits (m_1, m_2) as a message pair to its challenger and receives a ciphertext, CT, which it uses to continue its simulation. Note that \mathcal{A}' can submit all its message queries prior to submitting any function queries. Whenever it would need to run $\text{FE}_{i^*}.\text{KeyGen}$ on a circuit C , it submits C as a function query and uses the response of its challenger to continue the simulation. When \mathcal{A} terminates, \mathcal{A}' outputs the same value as \mathcal{A} . Note that \mathcal{A}' simulates both Hyb_1 and Hyb_2 exactly and is an admissible adversary since all computable function evaluations are identical across hybrids, so \mathcal{A}' wins if \mathcal{A} wins, contradicting the security of FE_{i^*} . \square

Lemma 6. *If CFHSS is a selectively-secure combiner-friendly homomorphic secret sharing scheme, then Hyb_2 and Hyb_3 are computationally indistinguishable.*

Proof. Suppose there exists an adversary \mathcal{A} that can distinguish between these two hybrids. Then, we construct an adversary \mathcal{A}' that can break the CFHSS security game. \mathcal{A}' begins by selecting i^* to be the secure index in the CFHSS security game. \mathcal{A}' then runs \mathcal{A} and simulates the role of the challenger. For every challenge message pair (x_0, x_1) submitted by \mathcal{A} , \mathcal{A}' queries its challenger on input query (x_0, x_1) to obtain input shares $\{s_{i,j,k}\}_{i,j,k \in [n] \setminus \{i^*\}}$. \mathcal{A}' then uses these shares and follows the procedure of Hyb_2 to generate a ciphertext, which it then sends to \mathcal{A} . Whenever \mathcal{A} submits a function query C , \mathcal{A}' queries its challenger on C to obtain function shares $\{C_{i,j,k}\}_{i,j,k \in [n]}$ and evaluations $\{C_{i,j,k}(s_{i,j,k})\}_{i,j,k \in [n]}$ and uses them to compute a function key SK_C according to Hyb_2 , which it then sends to \mathcal{A} . Note that when the bit $b' = 0$ for the CFHSS security game, then \mathcal{A}' simulates Hyb_2 exactly and when $b' = 1$, then \mathcal{A}' simulates Hyb_3 exactly. Furthermore,

\mathcal{A}' is an admissible adversary since \mathcal{A} is an admissible adversary. Therefore, \mathcal{A}' wins if \mathcal{A} wins, contradicting the security of CFHSS. \square

Lemma 7. *If FE_{i^*} is a selectively-secure FE scheme, then Hyb_3 and Hyb_4 are computationally indistinguishable.*

Proof. This follows from an analogous proof as that of [Lemma 5](#). \square

Lemma 8. *If E is a λ -bit CPA-secure secret-key encryption scheme, then Hyb_4 and Hyb_5 are computationally indistinguishable.*

Proof. This follows from an analogous proof as that of [Lemma 4](#). \square

Lemma 9. *Hyb_5 and Hyb_6 are indistinguishable.*

Proof. Hyb_5 and Hyb_6 are identical games, so they are indistinguishable. \square

So, it follows that Hyb_0 and Hyb_6 are computationally indistinguishable and, therefore, the real game with $b = 0$ is computationally indistinguishable from the real game with $b = 1$. \square

7 Robust FE Combiners and Universal FE

We can consider a stronger notion of an FE combiner called a *robust* FE combiner. A robust FE combiner is an FE combiner that satisfies correctness and security provided that at least one FE candidate, FE_i , satisfies both correctness and security. No restrictions are placed on the other FE candidates. In particular, they may satisfy neither correctness nor security. We note that the FE combiner constructed in [Section 6](#) is not robust. However, [\[ABJ⁺19\]](#) showed how to unconditionally transform an FE combiner into a robust FE combiner.

Theorem 10 ([\[ABJ⁺19\]](#)). *If there exists an FE combiner, then there exists a robust FE combiner.*

Combining [Theorem 10](#) with [Theorem 8](#), we obtain the following corollary.

Corollary 3. *There exists an unconditionally secure unbounded-key robust FE combiner for $n = \text{poly}(\lambda)$ FE candidates for P/poly.*

Universal Functional Encryption: Robust FE combiners are closely related to the notion of universal functional encryption. Universal functional encryption is a construction of functional encryption satisfying the following simple guarantee. If there exists a Turing Machine with running time bounded by some $T(n) = \text{poly}(n)$ that implements a correct and secure FE scheme, then the universal functional encryption construction is itself a correct and secure FE scheme. Using the existence of a robust FE combiner ([Corollary 3](#)) and the results of [\[AJN⁺16, ABJ⁺19\]](#), we obtain the following corollary.

Corollary 4. *There exists a universal unbounded-key functional encryption scheme for P/poly.*

8 Acknowledgements

We thank Saikrishna Badrinarayanan for helpful discussions and the anonymous EUROCRYPT reviewers for useful feedback regarding this work. The authors were supported in part by a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1619348, 1228984, 1136174, and 1065276, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. Aayush Jain was also supported by a Google PhD Fellowship (2018) in the area of Privacy and Security. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, the U.S. Government, or Google.

References

- [ABJ⁺19] Prabhanjan Ananth, Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. From fe combiners to secure mpc and back. In *TCC*, 2019.
- [ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In *CRYPTO*, 2015.
- [Agr19] Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In *EUROCRYPT*, 2019.
- [AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, 2015.
- [AJL⁺19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In *CRYPTO*, 2019.
- [AJN⁺16] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In *CRYPTO*, 2016.
- [AJS17] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Robust transforming combiners from indistinguishability obfuscation to functional encryption. In *EUROCRYPT*, 2017.
- [AJS18] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. *IACR Cryptology ePrint Archive*, 2018:615, 2018.
- [AS17] Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In *EUROCRYPT*, 2017.
- [BBKK18] Boaz Barak, Zvika Brakerski, Ilan Komargodski, and Praves Kothari. Limits on low-degree pseudorandom generators (or: Sum-of-squares meets program obfuscation). In *EUROCRYPT*, 2018.

- [BGG⁺18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M.R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, 2018.
- [BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT*, pages 337–367, 2015.
- [BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO*, 2016.
- [BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, pages 1292–1303, 2016.
- [BGJS17] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. A note on vrfs from verifiable functional encryption. *IACR Cryptology ePrint Archive*, 2017:51, 2017.
- [BHJ⁺19] Boaz Barak, Samuel Hopkins, Aayush Jain, Pravesh Kothari, and Amit Sahai. Sum-of-squares meets program obfuscation, revisited. In *EUROCRYPT*, 2019.
- [BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, 2012.
- [Bit17] Nir Bitansky. Verifiable random functions from non-interactive witness-indistinguishable proofs. In *TCC*, 2017.
- [BNPW16] Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. In *TCC Part II*, 2016.
- [BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, 2011.
- [BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.
- [BV16] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation: from approximate to exact. In *TCC*, 2016.
- [BV17] Nir Bitansky and Vinod Vaikuntanathan. A note on perfect correctness by derandomization. In *EUROCRYPT*, 2017.
- [CGH⁺15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrede Lepoint, Hemanta K Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New mmap attacks and their limitations. In *CRYPTO*, 2015.
- [CHL⁺15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, 2015.
- [CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for cspr problems and cryptanalysis of the ggh multilinear map without an encoding of zero. In *ANTS*, 2016.

- [CLLT16] Jean-Sebastien Coron, Moon Sung Lee, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of ggh15 multilinear maps. In *CRYPTO*, 2016.
- [FHNS16] Marc Fischlin, Amir Herzberg, Hod Bin Noon, and Haya Shulman. Obfuscation combiners. In *CRYPTO*, 2016.
- [FL07] Marc Fischlin and Anja Lehmann. Security-amplifying combiners for collision-resistant hash functions. In *CRYPTO*, 2007.
- [GGH⁺13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.
- [GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. *IACR Cryptology ePrint Archive*, 2014:666, 2014.
- [GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In *TCC*, 2017.
- [GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round mpc: Information-theoretic and black-box. In *TCC*, 2018.
- [GIS19] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Personal communication, 2019.
- [GKP⁺13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, pages 218–229. ACM, 1987.
- [GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In *CRYPTO*, 2016.
- [GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In *EUROCRYPT*, 2017.
- [GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. *EUROCRYPT*, 2018.
- [HIKN08] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. Ot-combiners via secure computation. In *TCC*, 2008.
- [HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of ggh map. In *EUROCRYPT*, 2016.
- [HJO⁺16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In *CRYPTO*, 2016.
- [HKN⁺05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In *EUROCRYPT*, 2005.

- [IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – efficiently. In *CRYPTO*, 2008.
- [KNT18] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. In *EUROCRYPT*, 2018.
- [KS17] Ilan Komargodski and Gil Segev. From minicrypt to obfustopia via private-key functional encryption. In *EUROCRYPT*, 2017.
- [Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In *EUROCRYPT*, 2016.
- [Lin17] Huijia Lin. Indistinguishability obfuscation from sxdh on 5-linear maps and locality-5 prgs. In *CRYPTO*, 2017.
- [LM18] Huijia Lin and Christian Matt. Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. *IACR Cryptology ePrint Archive*, 2018:646, 2018.
- [LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao’s protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, April 2009.
- [LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local prgs. In *CRYPTO*, 2017.
- [LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In *FOCS*, 2016.
- [MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key fhe. In *EUROCRYPT*, 2016.
- [O’N10] Adam O’Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.
- [SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.
- [Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

A Additional Preliminaries

A.1 Secure Multi-Party Computation

We now provide the necessary background on secure multi-party computation. We first present the syntax and then the security definition.

Syntax. We define a secure multi-party computation protocol Π for n parties P_1, \dots, P_n associated with an n -party functionality $f : \{0, 1\}^{\ell_1} \times \dots \times \{0, 1\}^{\ell_n} \rightarrow \{0, 1\}^{\ell_{y_1}} \times \dots \times \{0, 1\}^{\ell_{y_n}}$. We let ℓ_i denote the length of the i^{th} party's input and ℓ_{y_i} denote the length of the i^{th} party's output. In any given execution of the protocol, the i^{th} party receives as input $x_i \in \{0, 1\}^{\ell_i}$ and maintains a state state_i . All the parties jointly compute the functionality $f(x_1, \dots, x_n)$ by following the protocol. The protocol proceeds in rounds and in each round, every party broadcasts a message to the other parties. In the end, party P_i outputs out_i . The protocol is said to be correct if the joint distribution $(\text{out}_1, \dots, \text{out}_n)$ is statistically close to $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$.

Semi-Honest Adversaries. We consider weaker adversarial models where the adversaries follow the instructions of the protocol. That is, they receive their inputs from the environment, behave as prescribed by the protocol, and, finally, output their view of the protocol. Such types of adversaries are referred to as semi-honest adversaries.

We define semi-honest security below. Denote $\text{Real}_{f,S}^{\Pi}(x_1, \dots, x_n)$ to be the joint distribution over the outputs of all the parties along with the views of the parties indexed by the set S .

Definition 11 (Semi-Honest Security). *Consider an n -party functionality f as defined above. Fix a set of inputs (x_1, \dots, x_n) , where $x_i \in \{0, 1\}^{\ell_i}$ and let r_i be the randomness of the i^{th} party. Let Π be a n -party protocol implementing f . We say that Π satisfies **security against semi-honest adversaries** if for every subset of parties S , there exists a PPT simulator Sim such that:*

$$\{ (\{y_i\}_{i \notin S}, \text{Sim}(\{y_i\}_{i \in S}, \{x_i\}_{i \in S})) \}, \cong_c \{ \text{Real}_{f,S}^{\Pi}(x_1, \dots, x_n) \},$$

where y_i is the i^{th} output of $f(x_1, \dots, x_n)$. In particular, we say that Π is semi-honest secure.

A.2 Garbling Schemes

We recall the definition of garbling schemes [Yao86, BHR12].

Definition 12 (Garbling Schemes [Yao86, BHR12]). *A garbling scheme $\text{GC} = (\text{GrbC}, \text{EvalGC})$ defined for a class of circuits \mathcal{C} consists of the following polynomial time algorithms:*

- **Garbled Circuit Generation, $\text{GrbC}(1^\lambda, C)$:** *On input the security parameter 1^λ and circuit $C \in \mathcal{C}$ with ℓ input bits, it generates the garbled circuit \widehat{C} along with labels $\mathbf{k} = (\mathbf{k}_1, \dots, \mathbf{k}_\ell)$, where $\mathbf{k}_i = (k_i^0, k_i^1)$.*
- **Evaluation, $\text{EvalGC}(\widehat{C}, (k_1^{x_1}, \dots, k_\ell^{x_\ell}))$:** *On input garbled circuit \widehat{C} and labels $(k_1^{x_1}, \dots, k_\ell^{x_\ell})$, it generates the output out .*

It satisfies the following properties:

- **Correctness:** *For every circuit $C \in \mathcal{C}$ with input length ℓ , $x \in \{0, 1\}^\ell$, for every security parameter $\lambda \in \mathbb{N}$, it should hold that:*

$$\Pr \left[\begin{array}{l} (\widehat{C}, (\mathbf{k}_1, \dots, \mathbf{k}_\ell)) \leftarrow \text{GrbC}(1^\lambda, C) \\ C(x) \leftarrow \text{EvalGC}(\widehat{C}, (k_1^{x_1}, \dots, k_\ell^{x_\ell})) \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

- **Security:** *There exists a PPT simulator SimGC such that the following holds for every circuit $C \in \mathcal{C}$ of input length ℓ , $x \in \{0, 1\}^\ell$,*

$$\left(\widehat{C}, k_1^{x_1}, \dots, k_\ell^{x_\ell}\right) \cong_c \text{SimGC}(1^\lambda, \phi(C), C(x)),$$

where:

- $(\widehat{C}, (\mathbf{k}_1, \dots, \mathbf{k}_\ell)) \leftarrow \text{GrbC}(1^\lambda, C)$
- $\phi(C)$ is the topology of C .

Theorem 11 ([Yao86, LP09, BHR12]). *Assuming the existence of one-way functions, there exists a secure garbling scheme GC .*

B Protocol of [GS18, GIS18]

In this section, we summarize the 2-round MPC protocol of [GS18] in the correlated randomness model that is secure against up to $n - 1$ semi-honest corruptions, assuming one-way functions. Large portions of this section are taken essentially verbatim from [GS18, GIS18].

The protocol of [GS18, GIS18] is constructed by first taking a protocol Φ that is a conforming protocol and then compiling it into the final protocol. We first define the notion of a conforming protocol in the correlated randomness model.

B.1 Conforming Protocols [GS18, GIS18]

Let Φ be an MPC protocol for n parties P_1, \dots, P_n with inputs x_1, \dots, x_n , each of length ℓ_{in} , where all parties obtain the same output y . Let ℓ_i^{rand} denote the length of randomness needed by a party P_i in the execution of Φ and let $\ell_{i,j}^{\text{OT}}$ denote the length of correlated randomness needed between parties P_i and P_j . WLOG (by simply taking the maximum over all lengths), the length of randomness needed for any party or the length of correlated randomness needed between any two parties in the execution of Φ can be some fixed length ℓ_r . We can view Φ as a deterministic MPC protocol, where the input of a party P_i is $(x_i, r_{i,1}, \dots, r_{i,n}) \in \{0, 1\}^m$ for $m = \ell_{\text{in}} + n\ell_r$, where $r_{i,i}$ is the randomness used by P_i and $(r_{i,j}, r_{j,i})$ for $i \neq j$ is the correlated randomness tuple used between parties P_i and P_j . A conforming protocol is then defined by an OT generation phase, algorithm Pre , computation steps or actions ϕ_1, \dots, ϕ_T , and an algorithm Post . These, in turn, make up the OT generation, pre-processing, computation, and output phases of a conforming protocol. Formally, the phases are defined as follows (these definitions are adapted from [GS18, GIS18]).

- **OT Generation Phase:** For every instance of OT to be performed in the protocol, generate correlated randomness between the two parties that perform the OT.
- **Pre-processing Phase:** For each $i \in [n]$, party P_i computes

$$(u_i, w_i) \leftarrow \text{Pre}(i),$$

where Pre is a randomized algorithm that takes as input the index i of the party and outputs a strings $u_i \in \{0, 1\}^m$ and $w_i \in \{0, 1\}^{\ell'}$. P_i then sets

$$z_i := (x_i, r_{i,1}, \dots, r_{i,n}) \oplus u_i || 0^{\ell'}$$

and

$$v_i := 0^{(m+\ell') \cdot (i-1)} \| u_i \| w_i \| 0^{(m+\ell') \cdot (n-i)}.$$

P_i keeps v_i as secret information and broadcasts z_i to all other parties. Letting $\ell/n = m + \ell'$, note that $z_i \in \{0, 1\}^{\ell/n}$ and $v_i \in \{0, 1\}^\ell$.

- **Computation Phase:** For each $i \in [n]$, party P_i sets

$$\text{st}_i := (z_1 \| \dots \| z_n) \oplus v_i.$$

Then, for t from 1 to T , the parties proceed as follows.

- Parse action ϕ_t as (i, a, b, c) , where $i \in [n]$ and $a, b, c \in [\ell]$.
- Party P_i computes *one* NAND gate as

$$\text{st}_{i,c} = \text{NAND}(\text{st}_{i,a}, \text{st}_{i,b})$$

and broadcasts $\text{st}_{i,c} \oplus v_{i,c}$ to every other party, where $v_{i,c}$ denotes the c th bit of v_i .

- Every party P_j for $j \neq i$ updates $\text{st}_{j,c}$ to the bit value received from P_i .

We require that for all $t, t' \in [T]$ such that $t \neq t'$, we have that if $\phi_t = (\cdot, \cdot, \cdot, c)$ and $\phi_{t'} = (\cdot, \cdot, \cdot, c')$, then $c \neq c'$. Furthermore, if $\phi_t = (i, \cdot, \cdot, c)$, then $c \in \{(i-1)\ell/n + m + 1, \dots, i\ell/n\}$. Also, we denote $A_i \subseteq [T]$ to be the set of rounds in which party P_i sends a bit. That is, $A_i = \{t \in T \mid \phi(t) = (i, \cdot, \cdot, \cdot)\}$.

- **Output Phase:** Every party P_i computes $\text{Post}(\{z_j\}_{j \in [n]}, \text{trans})$, where trans is the transcript of the computation phase. Here, Post is an algorithm that takes as input the entire transcript of the protocol and computes the output of the MPC protocol.

B.2 Transforming an Arbitrary Protocol into a Conforming One

We now show how to transform any arbitrary MPC protocol Π with a publicly recoverable output in the correlated randomness model into a conforming protocol Φ . The transformation is the same as the one in [GS18, GIS18] for conforming protocols. The transformed protocol maintains the correctness and security of the underlying protocol.

Lemma 10 ([GS18, GIS18]). *Any MPC protocol Π in the correlated randomness model with a publicly recoverable output can be written as a conforming protocol Φ while inheriting the correctness and security of the original protocol.*

Proof. This was shown in [GS18, GIS18]. They define a conforming protocol slightly differently than we do, and we make these definitional changes in order to make input-locality easier to see later on. In the definition of a conforming protocol [GS18, GIS18], they define the pre-processing phase by an algorithm Pre , where $\text{Pre}(i, x_i)$ outputs strings (z_i, v_i) with $z_i \in \{0, 1\}^{\ell/n}$, $v_i \in \{0, 1\}^\ell$. However, in their transformation, they define Pre to exactly match our pre-processing phase. Simply have our $\text{Pre}(i)$ sample $u_i \leftarrow \{0, 1\}^m$ and $w_i \leftarrow (\{0, 1\}^{q-1} \| 0)^{p/n}$, for the parameters p, q mentioned in their transformation. The rest of their description of Pre exactly matches the definition of the pre-processing phase for a conforming protocol. Furthermore, the Post algorithm is to simply run the output algorithm of Π given the transcript of Π and a party's state. If the output is publicly

recoverable (it only depends on the transcript), then the Post algorithm given by [GS18] does not use a party's state and, thus, the local state of a party can be omitted from the syntax of Post as we have defined it above.

Remark 3. *The property that if $\phi_t = (i, \cdot, \cdot, c)$, then $c \in \{(i-1)l/n + m + 1, \dots, il/n\}$ is not explicitly written in the definition of a conforming protocol in [GS18, GIS18]. However, their transformation to a conforming protocol satisfies this property and it is required later in their proof of security. This simply means that when P_i sends a message, the bit is written to P_i 's part of the state.*

To instantiate Lemma 10, we can simply set Π to be the MPC protocol of [GMW87] in the correlated randomness model. In this work, we only care about protocols where all parties obtain the same output, and the protocol of [GMW87] has a publicly recoverable output.

B.3 Compiled Protocol

Here, we give the description of the 2-round MPC protocol of [GS18, GIS18] in the correlated randomness model.

Let Φ be an n -party conforming semi-honest MPC protocol (with T rounds in the computation phase) and GC be a garbling scheme.

Pre-processing Phase:

- For each $\alpha, \beta \in \{0, 1\}$:
 - For each $t \in [T]$, parse action ϕ_t as (i^*, a, b, c) . Generate an OT correlation between P_{i^*} and P_j for all other parties P_j where P_{i^*} is the OT receiver and P_j is the OT sender. For the OT correlation between P_{i^*} and P_j , let $(b_{t,j,\alpha,\beta}, r_{t,j,\alpha,\beta})$ refer to P_{i^*} 's correlated randomness and $(r_{t,j,\alpha,\beta}^0, r_{t,j,\alpha,\beta}^1)$ refer to P_j 's correlated randomness.
- Generate OT correlations needed to execute the conforming protocol Φ .

The description of the computation phase of the protocol is given in Figure 7.

Round 1: Each party P_i does the following:

- Compute $(u_i, w_i) \leftarrow \text{Pre}(i)$.
- Set

$$z_i := (x_i, r_{i,1}, \dots, r_{i,n}) \oplus u_i \parallel 0^{\ell'}$$

and

$$v_i := 0^{(m+\ell') \cdot (i-1)} \parallel u_i \parallel w_i \parallel 0^{(m+\ell') \cdot (n-i)}.$$

- For each $t \in A_i$ (recall the definition of A_i from the definition of a conforming protocol), parse $\phi_t = (i, a, b, c)$. For each $j \neq i \in [n]$ and $\alpha, \beta \in \{0, 1\}$, compute

$$\text{OT}_{t,j,\alpha,\beta}^1 := (v_{i,c} \oplus \text{NAND}(v_{i,a} \oplus \alpha, v_{i,b} \oplus \beta)) \oplus b_{t,j,\alpha,\beta}.$$

- Send $(z_i, \{\text{OT}_{t,j,\alpha,\beta}^1\}_{t \in A_i, j \neq i \in [n], \alpha, \beta \in \{0,1\}})$ to every other party.

Round 2: In the second round, each party P_i does the following:

- Set $\text{st}_i := (z_1 \parallel z_2 \parallel \dots \parallel z_n) \oplus v_i$.
- Set $\overline{\text{Label}}^{i,T+1} := \{\text{Label}_{k,0}^{i,T+1}, \text{Label}_{k,1}^{i,T+1}\}_{k \in [\ell]}$ where for each $k \in [\ell]$ and $b \in \{0, 1\}$, $\text{Label}_{k,b}^{i,T+1} := 0^\lambda$.
- For each t from T down to 1,

- Parse ϕ_t as (i^*, a, b, c) .
- If $i = i^*$ then compute (where **Talk** is described in [Figure 8](#))

$$\begin{aligned} (\widehat{\text{Talk}}_{i,t}, \overline{\text{Label}}^{i,t}) &\leftarrow \text{GrbC}(1^\lambda, \text{Talk}[i, \phi_t, v_i, \\ &\{(b_{t,j,\alpha,\beta}, r_{t,j,\alpha,\beta})\}_{j \neq i \in [n], \alpha, \beta \in \{0,1\}}, \overline{\text{Label}}^{i,t+1}]). \end{aligned}$$

- If $i \neq i^*$ then,
 - * For every $\alpha, \beta \in \{0, 1\}$, compute

$$\text{OT}_{t,i^*,\alpha,\beta}^2 := (\text{Label}_{c,0}^{i,t+1} \oplus X_0, \text{Label}_{c,1}^{i,t+1} \oplus X_1),$$

where $X_b = r_{t,i^*,\alpha,\beta}^{b \oplus \text{OT}_{t,i^*,\alpha,\beta}^1}$.

- * Compute (where **Listen** is described in [Figure 9](#))

$$\begin{aligned} (\widehat{\text{Listen}}_{i,t}, \overline{\text{Label}}^{i,t}) &\leftarrow \text{GrbC}(1^\lambda, \text{Listen}[i, \phi_t, \\ &\{\text{OT}_{t,i^*,\alpha,\beta}^2\}_{\alpha, \beta \in \{0,1\}}, \overline{\text{Label}}^{i,t+1}]). \end{aligned}$$

- Send $(\{\widehat{\text{Talk}}_{i,t}\}_{t \in A_i}, \{\widehat{\text{Listen}}_{i,t}\}_{t \in [T] \setminus A_i}, \{\text{Label}_{k,\text{st}_i,k}^{i,1}\}_{k \in [\ell]})$ to every other party.

Figure 7: Two-round MPC in the correlated randomness model [[GS18](#), [GIS18](#)].

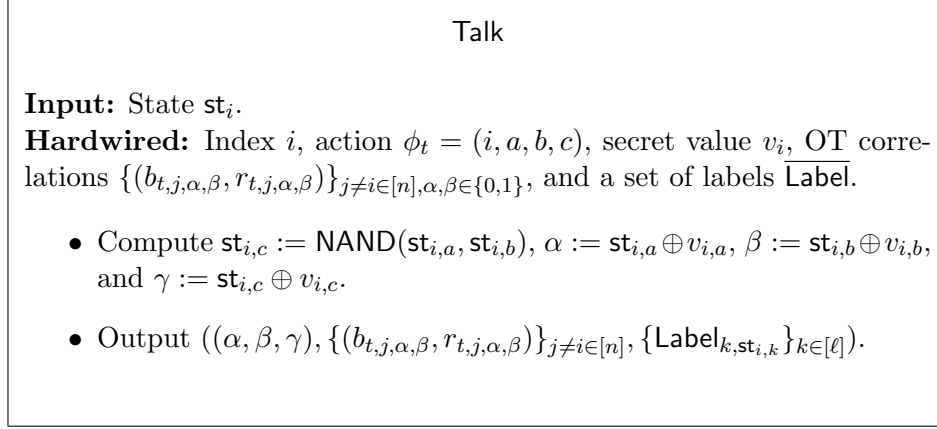


Figure 8: Description of Circuit Talk.

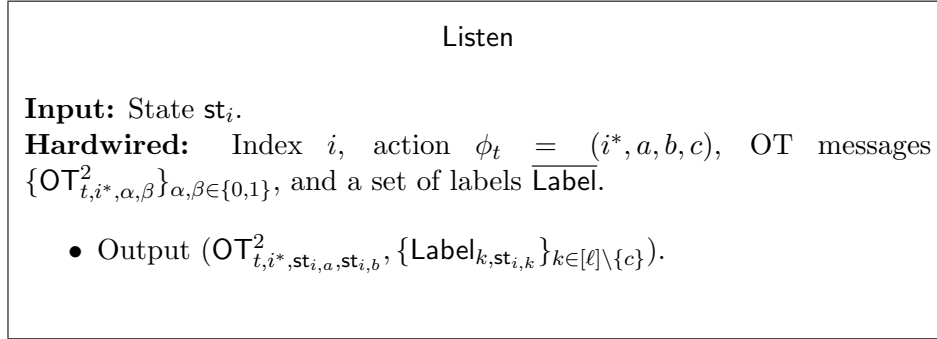


Figure 9: Description of Circuit Listen.

Evaluation. To compute the output of the protocol, do the following:

- Set $st := (z_1 || z_2 || \dots || z_n)$.
- For each $j \in [n]$, let $\widetilde{\text{Label}}^{j,1} := \{\widetilde{\text{Label}}_k^{j,1}\}_{k \in [\ell]}$ be the input labels received from party P_j at the end of round 2.
- For each t from 1 to T do:
 - Parse ϕ_t as (i^*, a, b, c) .
 - Compute $((\alpha, \beta, \gamma), \{(b_{t,j}, r_{t,j})\}_{j \in [n] \setminus \{i^*\}}, \widetilde{\text{Label}}^{i^*,t+1}) := \text{EvalGC}(\widetilde{\text{Talk}}_{i^*,t}, \widetilde{\text{Label}}^{i^*,t})$.
 - Set $st_h := \gamma$.
 - For each $j \neq i^*$ do:
 - * Compute $(\text{OT}^2, \{\text{Label}_k^{j,t+1}\}_{k \in [\ell] \setminus \{c\}}) := \text{EvalGC}(\widetilde{\text{Listen}}_{j,t}, \widetilde{\text{Label}}^{j,t})$.
 - * Parse OT^2 as (Y_0, Y_1) .
 - * Recover $\text{Label}_c^{j,t+1} := Y_{b_{t,j}} \oplus r_{t,j}$.
 - * Set $\widetilde{\text{Label}}^{j,t+1} := \{\text{Label}_k^{j,t+1}\}_{k \in [\ell]}$.

Compute the output as $\text{Post}(st)$.

Correctness and Security: Formal proofs of correctness and security can be found in [GS18, GIS18]. We provide some intuition below.

At a high level, correctness follows from the fact that given the transcript, the evaluator computes the output by maintaining a state \mathbf{st} and updating it for every action ϕ_t of the underlying conforming protocol Φ . At the beginning, the evaluator has input labels corresponding to the initial state of every party in an execution of Φ and garbled circuits which allow the evaluator to obtain input labels for the state of every party after each action in the underlying conforming protocol. Once the garbled circuits have been evaluated for every action ϕ_t , \mathbf{st} now contains the transcript of the underlying conforming protocol, from which the output can be determined.

For completeness, we give a description of the simulator Sim for the above protocol and refer the reader to [GS18, GIS18] for the proof of indistinguishability between the real execution and the output of the simulator. This description is taken virtually verbatim from [GS18]. Let Sim_Φ denote the simulator for the conforming protocol Φ and Sim_{GC} denote the simulator for the garbling scheme GC . Let \mathcal{A} be a semi-honest adversary and let $H \subseteq [n]$ be the set of honest/uncorrupted parties.

Sim is given the output y of the protocol and runs $\text{Sim}_\Phi(1^\lambda, \{x_i\}_{i \notin H}, y)$ to obtain $\{z_i\}_{i \in H}$, the random tapes of the corrupted parties, and the transcript of the computation phase of Φ denoted by $Z \in \{0, 1\}^T$, where Z_t is the bit sent in the t th round of the computation phase. Using the random tapes of the corrupted parties, Sim obtains $\{z_i\}_{i \notin H}$. Let $\mathbf{st}^* = z_1 || z_2 || \dots || z_n$. For every $t \in [T]$, parse $\phi_t = (i, (a, b, c))$ and update $\mathbf{st}_c^* = Z_c$. Sim starts the real-world adversary \mathcal{A} with inputs $\{z_i\}_{i \in H}$ and random tape generated by Sim_Φ .

Round-1 messages from Sim to \mathcal{A} : Sim generates correlated randomness to be used by honest parties (making it consistent with known correlated randomness of corrupted parties). For each $i \in H$, $t \in A_i$, $j \neq i \in [n]$, $\alpha, \beta \in \{0, 1\}$, it generates $\text{OT}_{t,j,\alpha,\beta}^1$ using choice bit Z_t and the appropriate correlated randomness. For each $i \in H$, Sim sends $(z_i, \{\text{OT}_{t,j,\alpha,\beta}^1\}_{t \in A_i, j \neq i \in [n], \alpha, \beta \in \{0, 1\}})$ to the adversary \mathcal{A} on behalf of the honest party P_i .

Round-1 messages from \mathcal{A} to Sim : For every $i \in [n] \setminus H$, Sim receives from the adversary \mathcal{A} the value $(z_i, \{\text{OT}_{t,j,\alpha,\beta}^1\}_{t \in A_i, j \neq i \in [n], \alpha, \beta \in \{0, 1\}})$ on behalf of the corrupted party P_i .

Round-2 messages from Sim to \mathcal{A} : For each $i \in H$, Sim generates the second round message on behalf of party P_i as follows:

- For each $k \in [\ell]$, set $\text{Label}_k^{i, T+1} := 0^\lambda$.
- For each t from T down to 1:
 - Parse ϕ_t as (i^*, a, b, c) .
 - Set $\alpha^* = \mathbf{st}_a^*$, $\beta^* = \mathbf{st}_b^*$, $\gamma^* = \mathbf{st}_c^*$.
 - If $i = i^*$, compute

$$\begin{aligned} (\widehat{\text{Talk}}_{i,t}, \text{Label}^{i,t}) &\leftarrow \text{Sim}_{\text{GC}}(1^\lambda, ((\alpha^*, \beta^*, \gamma^*), \\ &\{(b_{t,j,\alpha^*,\beta^*}, r_{t,j,\alpha^*,\beta^*})\}_{j \neq i \in [n]}, \{\text{Label}_k^{i,t+1}\}_{k \in [\ell]})). \end{aligned}$$

- If $i \neq i^*$, then set $\text{OT}_{t,i^*,\alpha^*,\beta^*}^2 := (\text{Label}_c^{i,t+1} \oplus X_0, \text{Label}_c^{i,t+1} \oplus X_1)$, where $X_b = r_{t,i^*,\alpha^*,\beta^*}^{b \oplus \text{OT}_{t,i,\alpha^*,\beta^*}^1}$. Then, compute

$$(\widehat{\text{Listen}}_{i,t}, \text{Label}^{i,t}) \leftarrow \text{Sim}_{\text{GC}}(1^\lambda, (\text{OT}_{t,i^*,\alpha^*,\beta^*}^2, \{\text{Label}_k^{i,t+1}\}_{k \in [\ell] \setminus \{c\}})).$$

- Send $(\{\widehat{\text{Talk}}_{i,t}\}_{t \in A_i}, \{\widehat{\text{Listen}}_{i,t}\}_{t \in [T] \setminus A_i}, \{\text{Label}_k^{i,1}\}_{k \in [\ell]})$ to every other party.

Round-2 messages from \mathcal{A} to Sim: For every $i \in [n] \setminus H$, Sim receives P_i 's round 2 message from \mathcal{A} .

C Input-Local MPC Protocol (Proof of Theorem 4)

In this section, we show that the MPC protocol of [GS18, GIS18] (see Appendix B), with the underlying garbling scheme instantiated as the one from Definition 7, is 3-input-local, proving Theorem 4.

Proof. We will show that the protocol Φ from Appendix B satisfies all the required properties of an input-local MPC protocol when the underlying garbling scheme is instantiated with the one from Definition 7. First, we observe that the evaluation procedure only requires knowledge of the transcript, so Φ satisfies the publicly recoverable output property. Furthermore, Φ is simulation secure against $n - 1$ semi-honest corruptions, assuming the existence of one-way functions (Theorem 2), so the required security property holds.

To see that Φ has an input-local transcript, we will go through every bit sent in an execution of Φ and note their dependencies on parties' inputs and randomness. In round 1, each party sends z_i and $\{\text{OT}_{t,j,\alpha,\beta}^1\}_{t \in A_i, j \neq i \in [n], \alpha, \beta \in \{0,1\}}$. Each bit of z_i depends on at most 2 parties (this occurs when the bit of z_i is the XOR of a correlated randomness bit of P_i and a bit of u_i). Furthermore, v_i only depends on the randomness of P_i and, therefore, each OT_1 message $\text{OT}_{t,j,\alpha,\beta}^1$ depends on at most 2 parties (P_i and P_j). In round 2, each party sends $(\{\widehat{\text{Talk}}_{i,t}\}_{t \in A_i}, \{\widehat{\text{Listen}}_{i,t}\}_{t \in [T] \setminus A_i}, \{\text{Label}_{k,\text{st}_i,k}^{i,1}\}_{k \in [\ell]})$ to every other party.

When using the garbling scheme of Definition 7, each $\widehat{\text{Talk}}_{i,t}$ consists of \hat{U} , a garbling of the universal circuit, and input labels corresponding to Talk and the string $(i, \phi_t, v_i, \{(b_{t,j,\alpha,\beta}, r_{t,j,\alpha,\beta})\}_{j \neq i \in [n], \alpha, \beta \in \{0,1\}}, \overline{\text{Label}}^{i,t+1})$. \hat{U} and the input labels corresponding to Talk only depend on the garbling party P_i . The bits of i, ϕ_t, v_i , and $\overline{\text{Label}}^{i,t+1}$ also only depend on P_i , while the bits of $(b_{t,j,\alpha,\beta}, r_{t,j,\alpha,\beta})$ depend on P_i and P_j . Thus, each bit of $\widehat{\text{Talk}}_{i,t}$ depends on at most 2 parties.

Similarly, each $\widehat{\text{Listen}}_{i,t}$ consists of \hat{U} and input labels corresponding to Listen and the string $(i, \phi_t, \{\text{OT}_{t,i^*,\alpha,\beta}^2\}_{\alpha,\beta \in \{0,1\}}, \overline{\text{Label}}^{i,t+1})$. \hat{U} and the input labels corresponding to Listen , i, ϕ_t , and $\overline{\text{Label}}^{i,t+1}$ only depend on the garbling party P_i . The bits of each $\text{OT}_{t,i^*,\alpha,\beta}^2$ depend on both P_i and P_{i^*} , so each bit of $\widehat{\text{Listen}}_{i,t}$ depends on at most 2 parties.

Finally, each bit of $\text{Label}_{k,\text{st}_i,k}^{i,1}$ is a bit of an input label corresponding to a bit of st_i . Each bit of st_i depends on P_i and at most 2 other parties (since each bit of z_j can depend on at most 2 parties) and, therefore, each bit of st_i depends on at most 3 parties. Therefore, we conclude that every bit in the transcript of Φ depends on at most 3 parties, and so Φ is 3-input-local. \square

D Proof of Theorem 9 [ABJ⁺19]

Proof. We observe that if d is a constant, the scheme described above is efficient if the underlying candidates are efficient. Correctness of FE_S follows immediately from the construction and the correctness of the underlying FE candidates.

Suppose that one of the FE candidates, say FE_j , is secure. Suppose that FE_S is not secure; that is, there exists an adversary \mathcal{A} that can break the security of FE_S . Then, consider the following adversary \mathcal{A}' that breaks the security of FE_j . \mathcal{A}' runs \mathcal{A} and simulates the challenger. \mathcal{A}' first runs $\text{FE}_i.\text{Setup}(1^\lambda)$ for all $i \in [n] \setminus \{j\}$ to obtain MSK_i 's. When \mathcal{A} submits a challenge message pair (m_0, m_1) , \mathcal{A}' computes $\text{CT}_{j-1,0}$ and $\text{CT}_{j-1,1}$ by encrypting m_0 and m_1 , respectively by repeated encryption under $\text{FE}_1, \dots, \text{FE}_{j-1}$. It then submits these two ciphertexts as its messages to its challenger and receives a ciphertext, CT_j . It then encrypts this ciphertext using $\text{FE}_{j+1}, \dots, \text{FE}_d$ to obtain CT_d , which it gives to \mathcal{A} as the ciphertext. When \mathcal{A} asks for a function key for a circuit C , \mathcal{A}' runs $\text{FE}_S.\text{KeyGen}$ on C until it needs to run the keygen algorithm for FE_j . Here, it queries its challenger for the appropriate function key and then uses this response to continue generating SK_C . It then gives the resulting function key SK_C to \mathcal{A} . When \mathcal{A} terminates, \mathcal{A}' outputs the same response. Note that \mathcal{A}' simulates the security game for \mathcal{A} perfectly and therefore wins whenever \mathcal{A} wins, contradicting the security of FE_j . Therefore, it follows that FE_S is a secure FE scheme. \square