# Combiners for Functional Encryption, Unconditionally

Aayush Jain, Nathan Manohar, and Amit Sahai

UCLA, Los Angeles, CA

**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 [39,28,38,42].

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 [5,6,2] (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 [52] and first formalized by [19,51], 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 $f_1, \ldots, f_q$, called *functional keys*, which can be used to learn the values $f_1(x), \ldots, f_q(x)$ given an encryption of $x$. Intuitively, the security notion states that the functional keys associated with $f_1, \ldots, f_q$ and an encryption of $x$ reveal nothing beyond the values $f_1(x), \ldots, f_q(x)$.

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 [34], universal samplers [34], reusable garbled circuits [36], verifiable random functions [37,13,10], and adaptive garbling [40]. FE has also helped improve our understanding of important theoretical questions, such as the hardness of Nash equilibrium [33,34]. One of the most important applications of FE is its implication to indistinguishabil-

ity obfuscation (iO for short), which is considered the holy grail of cryptography [8,15]. In fact, if we are willing to tolerate subexponential security loss, then even secret-key FE is enough to imply iO [14,44,43].

Over the past few years, many constructions of functional encryption have been proposed [29,30,45,49,9,46,48,7,1,47,4] and studying what assumptions suffice for constructing general-purpose FE remains a very important and active area of investigation. Recent cryptanalytic attacks [23,41,26,24,25,11,49,12] 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 [6,2]. 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 [2].
- Constructions of robust FE combiners have encouraged research on understanding correctness amplification for FE, iO [16,6], and other fundamental cryptographic primitives [17].
- 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 [7] 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 [5,27], both appearing at CRYPTO, that studied the related question of obfuscation combiners. This was followed up by the work of [6], 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 [2] 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* $\mathsf{P/poly}$.

It turns out that our construction of an FE combiner also gives rise to a robust FE combiner using the results of [6,2].

**Corollary 1 (Informal).** *There exists an unconditionally secure robust FE combiner for* $\mathsf{P/poly}$.

As any robust FE combiner gives a universal FE scheme [5,6], we obtain the following additional result.

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

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 [20,21,22,50,18]). Suppose for a function $f$, it was possible to give out function shares $f_1, \ldots, f_n$ such that for any input $x$, we can $n$-out-of-$n$ secret share $x$ into shares $x_1, \ldots, x_n$ and recover $f(x)$ given $f_1(x_1), \ldots, f_n(x_n)$. 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 $\mathsf{FE}_i$ (depicted in Figure 1). To generate a function key for a function $f$, $\mathsf{FE}_i$ 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 [6,2] 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.

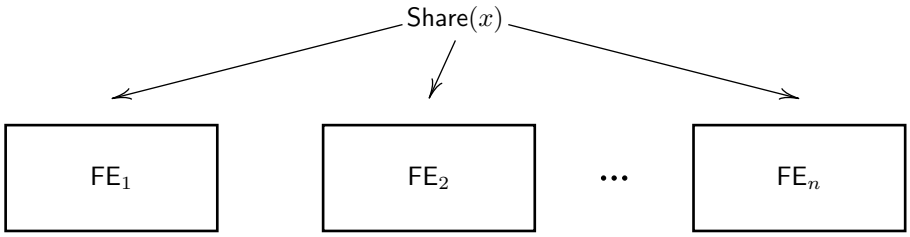


**Fig. 1.** A pictorial overview of splitting $x$ amongst $n$ FE candidates.

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, $\mathsf{FE}_1$ and $\mathsf{FE}_2$, we could combine these two candidates by simply having encryption encrypt first under $\mathsf{FE}_1$ and then encrypt the resulting ciphertext under $\mathsf{FE}_2$. To generate a secret key for a function $f$, we would generate a function key $\mathsf{SK}_{f,1}$ for $f$ under $\mathsf{FE}_1$ and then generate a function key $\mathsf{SK}_{f,2}$ for the function that runs $\mathsf{FE}_1.\mathsf{Dec}(\mathsf{SK}_{f,1}, \cdot)$ under $\mathsf{FE}_2$. The function key $\mathsf{SK}_{f,2}$ 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 2-nestings $\mathsf{FE}_{i,j}$ 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 $\mathsf{FE}_{i^*}$ was originally secure, then $\mathsf{FE}_{i,j}$ 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.

Suppose we had a "special" MPC protocol $\Phi$ where every bit in the transcript of an execution of $\Phi$ can be computed by a function on the inputs (and random coins) of a *constant* number of parties (say 2). Furthermore, the output of $\Phi$ can

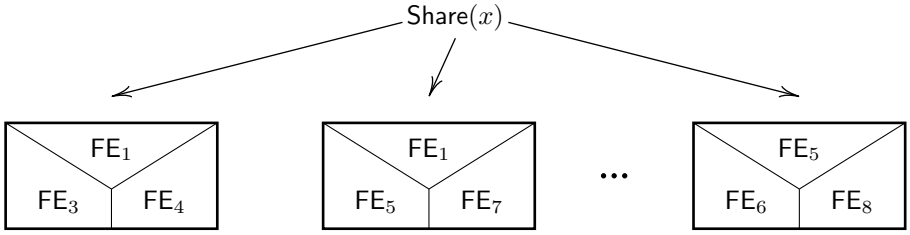


**Fig. 2.** A pictorial overview of 3-nested FE candidates (the required level of nesting in our construction). If $\mathsf{FE}_5$ is secure, then $\mathsf{FE}_{1,5,7}$ and $\mathsf{FE}_{5,6,8}$ are secure.

be determined solely from the transcript and $\Phi$ is secure against a semi-honest adversary that corrupts up to $n-1$ parties. If $\Phi$ has the above properties, then the transcript of an execution of $\Phi$ can be determined via an alternate computation. Instead of running $\Phi$ 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 $\tau_\alpha$ in the transcript $\tau$ 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 $\tau_\alpha$ in an execution of $\Phi$ 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 $(x_1, \ldots, x_n)$ and encrypt each pair of shares $(x_i, x_j)$ under $\mathsf{FE}_{i,j}$. To generate a function key for a function $f$, consider the MPC protocol that computes $f(x_1 \oplus \ldots \oplus x_n)$. Then, for every bit $\tau_\alpha$ in the transcript of such a protocol, if $\tau_\alpha$ "depends" on parties $P_i, P_j$, we would generate a function key under $\mathsf{FE}_{i,j}$ for the circuit $C_{\tau_\alpha}$ that computes $\tau_\alpha$ 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 $\mathsf{FE}_{i,j}$ and uses $K_i$ and some fixed tag $\mathsf{tag}_f$ 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 $C_{\tau_\alpha}$'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 $\mathsf{FE}_{i^*}$ 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 $\Phi$ where every bit in the transcript of an execution of $\Phi$ 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 $\Phi$ can be determined solely from the transcript and $\Phi$ 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 $\Phi$. 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 $\Phi$ 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 $\Phi$ 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 $\tau_\alpha$ in the transcript $\tau$ of an execution of $\Phi$ can be computed by some deterministic function $f_\alpha$ 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 [35] transformed to operate in the OT-hybrid model [31] 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 [35,31] 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 [35,31], 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 [35,31] 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 [3].

*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 [35,31], 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 $\lambda$. For an integer $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. 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 $\mathsf{negl}(\lambda)$ to denote a function that is negligible in $\lambda$. 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 $\mathsf{FE}$ for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four polynomial time algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ defined as follows. Let $\mathcal{X}_\lambda$ be the input space of the circuit class $\mathcal{C}_\lambda$ and let $\mathcal{Y}_\lambda$ be the output space of $\mathcal{C}_\lambda$. We refer to $\mathcal{X}_\lambda$ and $\mathcal{Y}_\lambda$ as the input and output space of the candidate/scheme, respectively.

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

– **Key Generation,** $\mathsf{SK}_C \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}, C)$: It takes as input the master secret key $\mathsf{MSK}$ and a circuit $C \in \mathcal{C}_\lambda$ and outputs a function key $\mathsf{SK}_C$.

– **Decryption,** $y \leftarrow \mathsf{FE.Dec}(\mathsf{SK}_C, \mathsf{CT})$: It takes as input a function secret key $\mathsf{SK}_C$, a ciphertext $\mathsf{CT}$ and outputs a value $y \in \mathcal{Y}_\lambda$.

Throughout this work, we will only be concerned with *uniform* algorithms. That is, $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ 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* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be correct if it satisfies the following property: for every* $C : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda \in \mathcal{C}_\lambda, m \in \mathcal{X}_\lambda$ *it holds that:*

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

*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 $\mathsf{Chal}$ and an adversary $\mathcal{A}$ where the adversary can request functional keys and ciphertexts from $\mathsf{Chal}$. Specifically, $\mathcal{A}$ can submit function queries $C$ and $\mathsf{Chal}$ 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* $\mathsf{FE}$ *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*

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

*where for each* $b \in \{0, 1\}$ *and* $\lambda \in \mathbb{N}$*, the experiment* $\mathsf{Expt}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, b)$ *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* $\mathsf{MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$ *and then computes* $\mathsf{CT}_i \leftarrow \mathsf{FE.Enc}(\mathsf{MSK}, x_b^i)$ *for all $i$. The challenger* Chal *then sends $\{\mathsf{CT}_i\}$ 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 $\mathsf{SK}_C \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}, C)$ 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 [3].

*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 the full version. In this work, we will deal with protocols that follow a certain structure, introduced in [35,31], called conforming protocols. The full syntactic definition of conforming protocols can be found in the full version.

### 2.3 Garbling Schemes

The definition of garbling schemes can be found in the full version.

### 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 $\mathsf{FEComb}$ takes $n$ FE candidates, $\mathsf{FE}_1, \ldots, \mathsf{FE}_n$ and compiles them into a new FE candidate with the property that $\mathsf{FEComb}$ 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 $\mathsf{FEComb}$ for a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four polynomial time algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ defined as follows. Let $\mathcal{X}_\lambda$ be the input space of the circuit class $\mathcal{C}_\lambda$ and let $\mathcal{Y}_\lambda$ be the output space of $\mathcal{C}_\lambda$. We refer to $\mathcal{X}_\lambda$ and $\mathcal{Y}_\lambda$ as the input and output space of the combiner, respectively. Furthermore, let $\mathsf{FE}_1, \ldots, \mathsf{FE}_n$ denote the descriptions of $n$ FE candidates.

- **Setup,** $\mathsf{FEComb.Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i \in [n]})$: It takes as input the security parameter $\lambda$ and the descriptions of $n$ FE candidates $\{\mathsf{FE}_i\}_{i \in [n]}$ and outputs the master secret key $\mathsf{MSK}$.

- **Encryption,** $\mathsf{FEComb.Enc}(\mathsf{MSK}, \{\mathsf{FE}_i\}_{i \in [n]}, m)$: It takes as input the master secret key $\mathsf{MSK}$, the descriptions of $n$ FE candidates $\{\mathsf{FE}_i\}_{i \in [n]}$, and a message $m \in \mathcal{X}_\lambda$ and outputs $\mathsf{CT}$, an encryption of $m$.

- **Key Generation,** $\mathsf{FEComb.Keygen}\left(\mathsf{MSK}, \{\mathsf{FE}_i\}_{i \in [n]}, C\right)$: It takes as input the master secret key $\mathsf{MSK}$, the descriptions of $n$ FE candidates $\{\mathsf{FE}_i\}_{i \in [n]}$, and a circuit $C \in \mathcal{C}_\lambda$ and outputs a function key $\mathsf{SK}_C$.

– **Decryption,** $\mathsf{FEComb.Dec}\left(\{\mathsf{FE}_i\}_{i\in[n]}, \mathsf{SK}_C, \mathsf{CT}\right)$: It is a deterministic algorithm that takes as input the descriptions of $n$ FE candidates $\{\mathsf{FE}_i\}_{i\in[n]}$, a function secret key $\mathsf{SK}_C$, and a ciphertext $\mathsf{CT}$ and outputs a value $y \in \mathcal{Y}_\lambda$.

*Remark 1.* In the formal definition above, we have included $\{\mathsf{FE}_i\}_{i\in[n]}$, the descriptions of the FE candidates, as input to all the algorithms of $\mathsf{FEComb}$. 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 $\mathsf{FEComb}$ are polynomial in $\lambda$ and $n$. Finally, security intuitively says that if at least one of the FE candidates is also secure, then $\mathsf{FEComb}$ is a secure FE scheme. We provide the formal definitions below.

*Correctness.*

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

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

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

*Polynomial Slowdown.*

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

*IND-Security.*

**Definition 5 (IND-Secure FE Combiner).** *An FE combiner $\mathsf{FEComb}$ is selectively secure if for any set $\{\mathsf{FE}_i\}_{i\in[n]}$ of correct FE candidates, it satisfies Definition 2, where the descriptions of $\{\mathsf{FE}_i\}_{i\in[n]}$ are public and implicit in all invocations of the algorithms of $\mathsf{FEComb}$,* if at least one *of the FE candidates $\mathsf{FE}_1, \ldots, \mathsf{FE}_n$ 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 $\mathsf{FEComb}$ 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 $\{\mathsf{FE}_i\}_{i\in[n]}$, 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 $\Pi_{\mathsf{univ}} = (\Pi_{\mathsf{univ}}.\mathsf{Setup}, \Pi_{\mathsf{univ}}.\mathsf{Enc}, \Pi_{\mathsf{univ}}.\mathsf{KeyGen}, \Pi_{\mathsf{univ}}.\mathsf{Dec})$ is a universal functional encryption scheme parametrized by $T$ if $\Pi_{\mathsf{univ}}$ 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 $\Phi$ be an MPC protocol for $n$ parties $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n$ in the correlated randomness model. We can view $\Phi$ 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 $P_j$. $\Phi$ is called $t$-input-local if the following holds:

- **Input-Local Transcript:** Let $\tau$ be a transcript of an execution of $\Phi$. Every bit $\tau_\alpha$ of $\tau$ 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 $f_\alpha$ such that

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

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

– **Publicly Recoverable Output:** Given a transcript $\tau$ of an execution of $\Phi$, there exists a function $\mathsf{Eval}$ such that the output of the protocol $\Phi$ for all parties is given by

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

– **Security:** $\Phi$ 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 [35,31] come "close", and we show that with a simple transformation, the protocol of [35,31] can be made $t$-input-local.

[35,31] show the following.

**Theorem 2 ([35],[31],[32]).** *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 [35] modified to operate in the correlated randomness model. In [31], they additionally modify the protocol of [35] in other ways, since the focus of [31] is on achieving information-theoretic security for smaller circuit classes and better efficiency. However, one can simply modify the protocol of [35] to operate in the correlated randomness model without making the additional modifications present in [31], a fact which we confirmed with the authors [32].

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 $\mathsf{GC} = (\mathsf{GrbC}, \mathsf{EvalGC})$ denote the standard Yao garbling scheme [53] for poly-sized circuits. Let $\mathcal{C}$ be a*

*class of keyed circuits with keyspace $\mathcal{V}$. Let the description length of any $C \in \mathcal{C}$ be $\ell_1$ and of any $v \in \mathcal{V}$ be $\ell_2$. Let the input length of any circuit $C \in \mathcal{C}$ be $\ell_3$. 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 $\mathsf{GC}' = (\mathsf{GrbC}', \mathsf{EvalGC}')$ be a garbling scheme for the class of keyed circuits $\mathcal{C}$ defined as follows:*

- **Garbled Circuit Generation, $\mathsf{GrbC}'(1^\lambda, C[v])$:** *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 $(\hat{U}, (\mathbf{k}_1, \ldots, \mathbf{k}_\ell)) \leftarrow \mathsf{GrbC}(1^\lambda, U)$. Output*

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

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

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

*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 $x \in \{0,1\}^{\ell_3}$, $\mathsf{EvalGC}'$ runs $\mathsf{EvalGC}$ 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 $\mathsf{SimGC}$ be the simulator for Yao's garbling scheme. The simulator $\mathsf{SimGC}'$ operates as follows. Run

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

and output

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

Suppose there exists an adversary $\mathcal{A}$ that can distinguish the output of $\mathsf{SimGC}'$ 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 $\mathsf{SimGC}'$, 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. □

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 included in the full version.

# 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,* $\mathsf{CFHSS} = (\mathsf{InpEncode}, \mathsf{FuncEncode}, \mathsf{Decode})$*, for a class of circuits* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *with input space* $\mathcal{X}_\lambda$ *and output space* $\mathcal{Y}_\lambda$ *supporting* $n \in \mathbb{N}$ *candidates consists of the following polynomial time algorithms:*

- **Input Encoding,** $\mathsf{InpEncode}(1^\lambda, 1^n, x)$*: It takes as input the security parameter* $\lambda$*, 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,** $\mathsf{FuncEncode}(1^\lambda, 1^n, C)$*: It is an algorithm that takes as input the security parameter* $\lambda$*, 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,** $\mathsf{Decode}(\{C_{i,j,k}(s_{i,j,k})\}_{i,j,k \in [n]})$*: 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,* $\mathsf{CFHSS}$*, 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:*

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

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

– **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*

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

*where for each* $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$ *and* $n \in \mathbb{N}$*, the experiment* $\mathsf{Expt}^{\mathsf{CFHSS}}_{\mathcal{A}}(1^\lambda, 1^n, b)$ *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* $L = \mathsf{poly}(\lambda)$ *is chosen by* $\mathcal{A}$*.*
3. *For all* $\ell$*,* Chal *computes* $\{s^\ell_{i,j,k}\}_{i,j,k\in[n]} \leftarrow \mathsf{InpEncode}(1^\lambda, 1^n, x_b^\ell)$*. For all* $\ell$*, the challenger* Chal *then sends* $\{s^\ell_{i,j,k}\}_{i,j,k\in[n]\setminus\{i^*\}}$*, 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* $\{C_{i,j,k}\}_{i,j,k\in[n]} \leftarrow \mathsf{FuncEncode}(1^\lambda, 1^n, C)$ *and sends them to* $\mathcal{A}$ *along with all evaluations* $\{C_{i,j,k}(s^\ell_{i,j,k})\}_{i,j,k\in[n]}$ *for all* $\ell \in [L]$*.*
5. *If there exists a function query* $C$ *and challenge message queries* $(x_0^\ell, x_1^\ell)$ *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 $\Phi_C$ is an MPC protocol for $C$ on $n$ parties if it computes the function $C(x_1 \oplus \ldots \oplus x_n)$ on inputs $x_1, \ldots, x_n$.

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* $n = \mathsf{poly}(\lambda)$ *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* P/poly *for* $n = \mathsf{poly}(\lambda)$ *candidates.*

*Notation:*

- Let $\mathsf{PRF}$ be a pseudorandom function with $\lambda$-bit keys that takes $\lambda$-bit inputs and outputs in $\{0,1\}^*$. $\mathsf{PRF}$ 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 $\mathsf{PRF}$ 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 $\mathsf{PRF}'$ be a pseudorandom function that maps $\{0,1\}^{2\lambda}$-bit inputs to a single output bit in $\{0,1\}$. Then, to evaluate $\mathsf{PRF}(K,x)$ to an appropriate output length $\ell$, we would simply compute the output bit by bit by evaluating $\mathsf{PRF}'(K,x||1), \mathsf{PRF}'(K,x||2), \ldots, \mathsf{PRF}'(K,x||\ell)$. When we write $(r_1,r_2,r_3) := \mathsf{PRF}(K,x)$, 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 $\mathsf{PRF}'(K,x||i||1), \mathsf{PRF}'(K,x||i||2), \ldots$ etc.
- For a 3-input-local protocol $\Phi$ 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 $\Phi$ for $C$ is included in the descriptions of the function shares for $C$.
- Let $\mathsf{Corr}(1^\lambda, 1^\ell, i, j) \to (r_{i,j}, r_{j,i})$ be the function that on input the security parameter $\lambda$, a length parameter $\ell$, 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, $\ell$ 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 $\ell$ in the description below when it is clear from the context. We note that $\mathsf{Corr}$ 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 $x_1, \ldots, x_n$, 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 $\Phi_C$ 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 $\tau$ of an execution of $\Phi_C$ and then running the evaluation procedure of $\Phi_C$.

*Construction:* We now provide the formal construction.

- **Input Encoding,** $\mathsf{InpEncode}(1^\lambda, 1^n, x)$:
  - XOR secret share $x$ uniformly at random across $n$ shares such that $x_1 \oplus \ldots \oplus x_n = x$.
  - 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,** $\mathsf{FuncEncode}(1^\lambda, 1^n, C)$: Let $\Phi$ denote the 3-input-local MPC protocol for $C$ on $n$ parties. For every bit $\tau_\alpha$ in $\tau$, a transcript of $\Phi$, let $\mathcal{S}_\alpha$ denote the set of parties that $\tau_\alpha$ depends on and $f_\alpha$ be the function that computes $\tau_\alpha$ with respect to these parties' inputs and randomness (see Section 4).
  - Sample tag $\mathsf{tag}_{\mathsf{rand}}$ 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.
  - Set all other function shares to $\perp$ and output $\{C_{i,j,k}\}_{i,j,k \in [n]}$.
- **Decoding,** $\mathsf{Decode}(\{C_{i,j,k}(s_{i,j,k})\}_{i,j,k \in [n]})$: It does the following:
  - Rearrange all input-local bits $\tau_\alpha$ output by the function shares to obtain $\tau$, the transcript of an execution of $\Phi$.
  - Run $\mathsf{Eval}(\tau)$ 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 $\mathsf{Decode}$ algorithm obtains $\tau$, the transcript of an execution of $\phi_C$. Therefore, by running $\mathsf{Eval}$ on $\tau$, $\mathsf{Decode}$ obtains

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

as desired.

$C_i$

**Input:** Input $x_i$ and PRF key $K_i$.
**Hardwired:** Index $i$, tag $\mathsf{tag}_{\mathsf{rand}}$ in $\{0,1\}^\lambda$.

- Compute $r_i := \mathsf{PRF}(K_i, \mathsf{tag}_{\mathsf{rand}})$.
- For every input-local bit $\tau_\alpha$ in a transcript $\tau$ of $\Phi$ with $\mathcal{S}_\alpha = \{i\}$, compute $\tau_\alpha := f_\alpha(x_i, r_i)$.
- Output $(\tau_\alpha)_{\tau_\alpha \text{ input-local with } \mathcal{S}_\alpha = \{i\}}$.

**Fig. 3.** Description of Function Share $C_i$.

$C_{i,j}$

**Input:** Inputs $x_i$, $x_j$ and PRF keys $K_i, K_j, K_{ij}$.
**Hardwired:** Indices $i, j$, tag $\mathsf{tag}_{\mathsf{rand}}$ in $\{0,1\}^\lambda$.

- For $u \in \{i, j\}$, compute $r_u := \mathsf{PRF}(K_u, \mathsf{tag}_{\mathsf{rand}})$.
- Compute $r_{ij}^{\mathsf{Corr}} := \mathsf{PRF}(K_{ij}, \mathsf{tag}_{\mathsf{rand}})$.
- Compute $(r_{i,j}, r_{j,i}) := \mathsf{Corr}(1^\lambda, i, j; r_{ij}^{\mathsf{Corr}})$.
- For every bit input-local bit $\tau_\alpha$ in a transcript $\tau$ of $\Phi$ with $\mathcal{S}_\alpha = \{i, j\}$, compute
$$\tau_\alpha := f_\alpha((x_u)_{u \in \mathcal{S}_\alpha}, (r_u)_{u \in \mathcal{S}_\alpha}, (r_{u,v})_{u,v \in \mathcal{S}_\alpha}).$$
- Output $(\tau_\alpha)_{\tau_\alpha \text{ input-local with } \mathcal{S}_\alpha = \{i,j\}}$.

**Fig. 4.** Description of Function Share $C_{i,j}$.

*Security:* The security proof can be found in the full version.

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

In the rest of this section, we show Theorem 7.

$C_{i,j,k}$

**Input:** Inputs $x_i, x_j, x_k$ and PRF keys $K_i, K_j, K_k, K_{ij}, K_{ik}, K_{jk}$.
**Hardwired:** Indices $i, j, k$, tag $\mathsf{tag}_{\mathsf{rand}}$ in $\{0,1\}^\lambda$.

- For $u \in \{i, j, k\}$, compute $r_u := \mathsf{PRF}(K_u, \mathsf{tag}_{\mathsf{rand}})$.
- Compute $r_{ij}^{\mathsf{Corr}} := \mathsf{PRF}(K_{ij}, \mathsf{tag}_{\mathsf{rand}})$, $r_{ik}^{\mathsf{Corr}} := \mathsf{PRF}(K_{ik}, \mathsf{tag}_{\mathsf{rand}})$, and $r_{jk}^{\mathsf{Corr}} := \mathsf{PRF}(K_{jk}, \mathsf{tag}_{\mathsf{rand}})$.
- Compute $(r_{i,j}, r_{j,i}) := \mathsf{Corr}(1^\lambda, i, j; r_{ij}^{\mathsf{Corr}})$, $(r_{i,k}, r_{k,i}) := \mathsf{Corr}(1^\lambda, i, k; r_{ik}^{\mathsf{Corr}})$, and $(r_{j,k}, r_{k,j}) := \mathsf{Corr}(1^\lambda, j, k; r_{jk}^{\mathsf{Corr}})$.
- For every bit input-local bit $\tau_\alpha$ in a transcript $\tau$ of $\Phi$ with $\mathcal{S}_\alpha = \{i, j, k\}$, compute
$$\tau_\alpha := f_\alpha((x_u)_{u \in \mathcal{S}_\alpha}, (r_u)_{u \in \mathcal{S}_\alpha}, (r_{u,v})_{u,v \in \mathcal{S}_\alpha}).$$
- Output $(\tau_\alpha)_{\tau_\alpha \text{ input-local with } \mathcal{S}_\alpha = \{i,j,k\}}$.

**Fig. 5.** Description of Circuit $C_{i,j,k}$.

### 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 $d$. The construction and proof of $d$-nested FE can be found in the full version.

### 6.2 Construction

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

*Notation:*

- Let $\mathsf{FE}_1, \ldots, \mathsf{FE}_n$ denote $n$ FE candidates. In the following construction, we assume that the descriptions $\{\mathsf{FE}_i\}_{i \in [n]}$ are implicit in all the algorithms of $\mathsf{FEComb}$.
- Let $\mathsf{FE}_{ijk}$ denote the 3-nested FE candidate derived by nesting $\mathsf{FE}_i$, $\mathsf{FE}_j$, and $\mathsf{FE}_k$.
- Let $\mathsf{CFHSS} = (\mathsf{InpEncode}, \mathsf{FuncEncode}, \mathsf{Decode})$ be a combiner-friendly homomorphic secret sharing scheme. Let $\ell_{\text{output}}$ denote the length of the outputs obtained from the evaluation of function shares on input shares.

– Let $\mathsf{E}$ be any $\lambda$-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 $\ell_{\mathsf{E}} = \ell_{\mathsf{E}}(\lambda)$ denote the length of the encryption key for the scheme $\mathsf{E}$.

*Construction:*

– $\mathsf{FEComb.Setup}(1^\lambda)$ : On input the security parameter, it runs $\mathsf{FE}_{ijk}.\mathsf{Setup}(1^\lambda)$ for $i,j,k \in [n]$ and $\mathsf{E.SK} \leftarrow \mathsf{E.Setup}(1^\lambda)$. It outputs $\mathsf{MSK} = (\{\mathsf{MSK}_{ijk}\}_{i,j,k\in[n]}, \mathsf{E.SK})$.

– $\mathsf{FEComb.Enc}(\mathsf{MSK}, x \in \{0,1\}^{\ell_x})$ : It executes the following steps.
  • First, encode $x$ into $n^3$ shares by running $\mathsf{CFHSS.InpEncode}(1^\lambda, 1^n, x)$ to compute $\{s_{i,j,k}\}_{i,j,k\in[n]}$. Then, for all $i,j,k \in [n]$, compute

$$\mathsf{CT}_{ijk} = \mathsf{FE}_{ij}.\mathsf{Enc}\left(\mathsf{MSK}_{ijk}, (s_{i,j,k}, 0^{\ell_{\mathsf{E}}}, 0)\right).$$

  • Output $\mathsf{CT} = \{\mathsf{CT}_{ijk}\}_{i,j,k\in[n]}$.

– $\mathsf{FEComb.KeyGen}(\mathsf{MSK}, C)$ : It executes the following steps.
  • For all $i,j,k \in [n]$, it computes $c_{i,j,k} \leftarrow \mathsf{E.Enc}(\mathsf{E.SK}, 0^{\ell_{\text{output}}})$, where $\ell_{\text{output}}$ is the length of evaluations of function shares on input shares of $\mathsf{CFHSS}$.
  • It computes $\{C_{i,j,k}\}_{i,j,k\in[n]} \leftarrow \mathsf{CFHSS.FuncEncode}(1^\lambda, 1^n, C)$.
  • For all $i,j,k \in [n]$, it computes $\mathsf{SK}_{H_{i,j,k}} \leftarrow \mathsf{FE}_{ijk}.\mathsf{KeyGen}\left(\mathsf{MSK}_{ijk}, H_{i,j,k}\right)$, where circuit $H_{i,j,k}$ is described in Figure 6.
  • It outputs $\mathsf{SK}_C = (\{\mathsf{SK}_{H_{i,j,k}}\}_{i,j,k\in[n]})$.

$H_{i,j,k}$

**Input:** Input share $s_{i,j,k}$, a string $t \in \{0,1\}^{\ell_{\mathsf{E}}}$, and a bit $b$
**Hardwired:** Ciphertext $c_{i,j,k}$, circuit $C_{i,j,k}$
∗ If $b \neq 0$, output $\mathsf{E.Dec}(t, c_{i,j,k})$.
∗ Otherwise, output $C_{i,j,k}(s_{i,j,k})$.

**Fig. 6.** Description of the Evaluation Circuit.

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

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

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

### 6.3 Security Proof

The security proof can be found in the full version.

## 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, $\mathsf{FE}_i$, 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, [2] showed how to unconditionally transform an FE combiner into a robust FE combiner.

**Theorem 8 ([2]).** *If there exists an FE combiner, then there exists a robust FE combiner.*

Combining Theorem 8 with Theorem 7, we obtain the following corollary.

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

*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 $T(n) = \mathsf{poly}(n)$ 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 [5,2], we obtain the following corollary.

**Corollary 4.** *There exists a universal unbounded-key functional encryption scheme for $\mathsf{P}/\mathsf{poly}$.*

## 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

1. Agrawal, S.: Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In: EUROCRYPT (2019)
2. Ananth, P., Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: From fe combiners to secure mpc and back. In: TCC (2019)
3. Ananth, P., Brakerski, Z., Segev, G., Vaikuntanathan, V.: From selective to adaptive security in functional encryption. In: CRYPTO (2015)
4. Ananth, P., Jain, A., Lin, H., Matt, C., Sahai, A.: Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In: CRYPTO (2019)
5. Ananth, P., Jain, A., Naor, M., Sahai, A., Yogev, E.: Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In: CRYPTO (2016)
6. Ananth, P., Jain, A., Sahai, A.: Robust transforming combiners from indistinguishability obfuscation to functional encryption. In: EUROCRYPT (2017)
7. Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. IACR Cryptology ePrint Archive 2018, 615 (2018)
8. Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: CRYPTO (2015)
9. Ananth, P., Sahai, A.: Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In: EUROCRYPT (2017)
10. Badrinarayanan, S., Goyal, V., Jain, A., Sahai, A.: A note on vrfs from verifiable functional encryption. IACR Cryptology ePrint Archive 2017, 51 (2017)
11. Barak, B., Brakerski, Z., Komargodski, I., Kothari, P.: Limits on low-degree pseudorandom generators (or: Sum-of-squares meets program obfuscation). In: EUROCRYPT (2018)
12. Barak, B., Hopkins, S., Jain, A., Kothari, P., Sahai, A.: Sum-of-squares meets program obfuscation, revisited. In: EUROCRYPT (2019)
13. Bitansky, N.: Verifiable random functions from non-interactive witness-indistinguishable proofs. In: TCC (2017)

14. Bitansky, N., Nishimaki, R., Passelègue, A., Wichs, D.: From cryptomania to obfustopia through secret-key functional encryption. In: TCC Part II (2016)
15. Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. In: FOCS (2015)
16. Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation: from approximate to exact. In: TCC (2016)
17. Bitansky, N., Vaikuntanathan, V.: A note on perfect correctness by derandomization. In: EUROCRYPT (2017)
18. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: CRYPTO (2018)
19. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: TCC (2011)
20. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: EUROCRYPT. pp. 337–367 (2015)
21. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: CRYPTO (2016)
22. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS. pp. 1292–1303 (2016)
23. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the multilinear map over the integers. In: EUROCRYPT (2015)
24. Cheon, J.H., Jeong, J., Lee, C.: An algorithm for cspr problems and cryptanalysis of the ggh multilinear map without an encoding of zero. In: ANTS (2016)
25. Coron, J.S., Gentry, C., Halevi, S., Lepoint, T., Maji, H.K., Miles, E., Raykova, M., Sahai, A., Tibouchi, M.: Zeroizing without low-level zeroes: New mmap attacks and their limitations. In: CRYPTO (2015)
26. Coron, J.S., Lee, M.S., Lepoint, T., Tibouchi, M.: Cryptanalysis of ggh15 multilinear maps. In: CRYPTO (2016)
27. Fischlin, M., Herzberg, A., Noon, H.B., Shulman, H.: Obfuscation combiners. In: CRYPTO (2016)
28. Fischlin, M., Lehmann, A.: Security-amplifying combiners for collision-resistant hash functions. In: CRYPTO (2007)
29. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS (2013)
30. Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Fully secure functional encryption without obfuscation. IACR Cryptology ePrint Archive 2014, 666 (2014)
31. Garg, S., Ishai, Y., Srinivasan, A.: Two-round mpc: Information-theoretic and black-box. In: TCC (2018)
32. Garg, S., Ishai, Y., Srinivasan, A.: Personal communication (2019)
33. Garg, S., Pandey, O., Srinivasan, A.: Revisiting the cryptographic hardness of finding a nash equilibrium. In: CRYPTO (2016)
34. Garg, S., Pandey, O., Srinivasan, A., Zhandry, M.: Breaking the sub-exponential barrier in obfustopia. In: EUROCRYPT (2017)
35. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. EUROCRYPT (2018)
36. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: STOC (2013)
37. Goyal, R., Hohenberger, S., Koppula, V., Waters, B.: A generic approach to constructing and proving verifiable random functions. In: TCC (2017)

38. Harnik, D., Ishai, Y., Kushilevitz, E., Nielsen, J.B.: Ot-combiners via secure computation. In: TCC (2008)
39. Harnik, D., Kilian, J., Naor, M., Reingold, O., Rosen, A.: On robust combiners for oblivious transfer and other primitives. In: EUROCRYPT (2005)
40. Hemenway, B., Jafargholi, Z., Ostrovsky, R., Scafuro, A., Wichs, D.: Adaptively secure garbled circuits from one-way functions. In: CRYPTO (2016)
41. Hu, Y., Jia, H.: Cryptanalysis of ggh map. In: EUROCRYPT (2016)
42. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: CRYPTO (2008)
43. Kitagawa, F., Nishimaki, R., Tanaka, K.: Obfustopia built on secret-key functional encryption. In: EUROCRYPT (2018)
44. Komargodski, I., Segev, G.: From minicrypt to obfustopia via private-key functional encryption. In: EUROCRYPT (2017)
45. Lin, H.: Indistinguishability obfuscation from constant-degree graded encoding schemes. In: EUROCRYPT (2016)
46. Lin, H.: Indistinguishability obfuscation from sxdh on 5-linear maps and locality-5 prgs. In: CRYPTO (2017)
47. Lin, H., Matt, C.: Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. IACR Cryptology ePrint Archive 2018, 646 (2018)
48. Lin, H., Tessaro, S.: Indistinguishability obfuscation from trilinear maps and blockwise local prgs. In: CRYPTO (2017)
49. Lin, H., Vaikuntanathan, V.: Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In: FOCS (2016)
50. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key fhe. In: EUROCRYPT (2016)
51. O'Neill, A.: Definitional issues in functional encryption. IACR Cryptology ePrint Archive 2010, 556 (2010)
52. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT (2005)
53. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS. pp. 162–167 (1986)