Round-Optimal and Communication-Efficient Multiparty Computation

Michele Ciampi[1], Rafail Ostrovsky[2], Hendrik Waldner[1], and Vassilis Zikas[1]

[1] The University of Edinburgh, Edinburgh, UK
{mciampi,hendrik.waldner}@ed.ac.uk,vzikas@inf.ed.ac.uk
[2] University of California, Los Angeles, CA, US
rafail@cs.ucla.edu

Abstract. Typical approaches for minimizing the round complexity of multi-party computation (MPC) do so at the cost of increased communication complexity (CC) or reliance on setup assumptions. A notable exception is the recent work of Ananth *et al.* [TCC 2019], which used Functional Encryption (FE) combiners to obtain a round optimal (two-round) semi-honest MPC in the plain model with CC proportional to the depth and input-output length of the circuit being computed—we refer to such protocols as *circuit scalable.* This leaves open the question of obtaining communication efficient *malicious* security in the plain model which we answer in this work:

1) We provide a round-preserving black-box compiler that compiles a wide class of MPC protocols into a *circuit-scalable* maliciously secure MPC in the plain model, assuming a (succinct) FE combiner. By using our compiler with a round-optimal MPC, we derive the first round-optimal and circuit-scalable maliciously secure MPC in the plain model.

2) We provide a round-preserving black-box compiler that compiles a wide class of MPC protocols into a *circuit-independent*— i.e., with CC that depends only on the input-output length of the circuit—maliciously secure MPC in the plain model, assuming Multi-Key Fully-Homomorphic Encryption (MFHE). Again, by using this second compiler with a round-optimal MPC, we derive the first round-optimal and circuit-independent maliciously secure MPC in the plain model. This is the best to-date CC for a round-optimal malicious MPC protocol, which is even communication-optimal when the output size of the function being evaluated is smaller than its input size (e.g., for boolean functions).

Our compilers assume the existence of four-round maliciously secure oblivious transfer which can be obtained from standard cryptographic assumptions.

1 Introduction

Secure multiparty computation (MPC) [Yao86, GMW87] allows different parties to jointly evaluate any circuit over private inputs in such a way that each party learns the output of the computation and nothing else. Many improvements in this area have led to better protocols in terms of complexity assumptions and round complexity in the case of malicious adversaries[3] [GMW87, Kil88, IPS08, GMW87, BMR90, KOS03, KO04, Pas04, PW10, Wee10, Goy11].

Recently, the design of round-optimal MPC has attracted a lot of attention. Concretely, for semi-honest adversaries, two rounds are necessary for secure MPC in the plain model (as any one-round protocol is trivially broken). A lower bound was matched by [BL18, GS18], where the authors present a two-round MPC protocol in the semi-honest model. Note that the above lower bound holds even assuming a correlated-randomness setup. Similarly, [BL18, GS18] shows that the same bound holds even for maliciously secure MPC, assuming a trusted correlated-randomness setup. However, Garg et al. [GMPP16] proved that in the plain model four rounds are necessary for maliciously secure MPC with a black-box simulator. This four-round lower-bound was matched by several constructions for a range of common (polynomial) complexity assumptions [CCG$^+$19, BGJ$^+$18, HHPV18]. Notwithstanding, a common drawback in all the above constructions is that their communication complexity is proportional to the size (of the description) of the circuit being evaluated. So the main question remains:

Is there a round-optimal MPC protocol in the plain model from standard complexity assumptions which is also *circuit-scalable*, i.e., has communication complexity that depends only the depth of the circuit being evaluated and its input and output length (and of course the security parameter)?

Note that assuming correlated randomness, the above question has been answered to the affirmative for malicious security by Quach et al. [QWW18], who proved that under the learning with errors assumption (LWE) it is possible to design a two-round circuit-scalable MPC protocol using a correlated-randomness setup. Moreover, the work of Morgan et al. [MPP20] shows that it is possible to construct a circuit-independent[4] 2-party computation protocol in which only one party gets the output relying on LWE only.[5] A first step into answering the above question in the plain model was taken for semi-honest adversaries in [ABJ$^+$19, QWW18]. Interesting, and most related to our results, Ananth et al. [ABJ$^+$19] achieved its round-optimal (two-round) and scalable solution by leveraging a connection between round-optimal semi-honest MPC and *functional encryption combiners*. However, their two-round construction does not work in the malicious setting, where, as already mentioned, four-rounds are necessary. This left the following important question open:

> *Is there a round-optimal and* circuit-scalable *maliciously secure MPC protocol in the plain model from standard (polynomial) complexity assumptions?*

As the first of our two main contributions, we answer the above question to the affirmative, by extending the investigation of the relation between FE combiners and MPC to the malicious setting. This completing the landscape of circuit-scalable and round optimal MPC in the plain

[3] A malicious adversary attacks the protocol following an arbitrary probabilistic polynomial-time strategy. Unless stated differently, when we talk about the security of an MPC protocol against semi-honest or malicious adversaries we assume that up to $n-1$ parties can be corrupted, where n is the number of parties.

[4] We stress that in our work the size of the circuit is always related via a polynomial p to the security parameter. We use the term circuit-independent for MPC protocols whose communication complexity depend on the security paramenters, the size of the input and the output, and does not depends on p. The same argument holds for circuit-scalable MPC protocols.

[5] The protocol proposed in [MPP20] does not assume broadcast, hence they obtain the best possible security guarantees in such a model.

model. More concretely, we provide a round-preserving black-box compiler that compiles a wide class of MPC protocols into a circuit-scalable protocol assuming any *succinct* FE combiner (see below). Note that such FE combiners are known to exist based on the learning with errors assumption. We next investigate whether our result can be strengthened to achieve *circuit-independent* MPC:

> *Is there a round-optimal and* circuit-independent *maliciously secure MPC protocol in the plain model from standard (polynomial) complexity assumptions?*

Although the connection between MPC and FE does not seem to help here, we still answer also the above question to the affirmative. Concretely, we propose a round-preserving black-box compiler that compiles a wide class of MPC protocols into a circuit-independent protocol assuming the existence of any *compact* Multi-Key Fully-Homomorphic Encryption (MFHE) scheme that enjoys perfect correctness. Informally, the compactness property, here, requires that the size of the cipthertexts and the size of the description of the encryption and decryption algorithms depends only on the input-output size of the function being computed.

For the special case of constant parties, the MFHE scheme required for our compiler exists based on perfect correct FHE [LTV12], which in turn can be instantiated from the LWE assumption [BGV12]. Hence our result yields the first circuit-independent round-optimal malicious MPC in the plain model for a constant number of parties—in particular the first such 2-PC—based on standard polynomial-time assumptions. For the case of arbitrary many parties, to our knowledge, such compact MFHE is only known to exist based on the Ring-LWE and the Decisional Small Polynomial Ratio (DSPR) assumption [LTV12]. Hence, under these assumptions, we obtain a circuit-independent round-optimal MPC protocol, as above, for arbitrary many parties. Deriving compact MFHE—and hence also a circuit-independent round-optimal MPC—from standard polynomial-time assumptions is an interesting open problem.

1.1 Related Work

Functional encryption (FE) [SW05, BSW11, O'N10] is a primitive that enables fine-grained access control of encrypted data. In more detail, a FE scheme is equipped with a key generation algorithm that allows the owner of a master secret key to generate a *secret key* sk_f associated with a circuit f. Using such a secret key sk_f for the decryption of a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, x)$ yields *only* $f(x)$. In other terms, the security of a functional encryption scheme guarantees that no other information except for $f(x)$ is leaked. A functional encryption combiner allows for the combination of many FE candidates in such a way that the resulting FE protocol is secure as long as any one of the initial FE candidates is secure. Ananth et al. [ABJ+19] show how to construct an FE combiner, based on the learning with errors (LWE) assumption, that enjoys the property of *succinctness* and *decomposability* (we elaborate more on the latter property in the next section). The property of succinctness states that 1) the length of each secret key is related to the depth and the length of the output of the circuit being evaluated and 2) the encryption complexity is proportional to the depth of the circuit being evaluated and to the length of the message being encrypted.

Given such a succinct FE combiner and an ℓ-round semi-honest MPC (not necessarily communication efficient), Ananth et al. show how to obtain an ℓ-round *circuit-scalable* MPC protocol that is secure against semi-honest adversaries. Given that such a combiner —as well as a round optimal semi-honest MPC— can be constructed from LWE, this result can be instantiated from the LWE assumption.

In [AJJM20] the authors also explore the relation between MFHE and MPC and, among other results, the authors show how to obtain a circuit-independent MPC protocol that is secure against semi-malicious adversary assuming Ring LWE, DSPR and 2-round OT[6].

1.2 Overview of our Results

From FE combiners to circuit-scalable MPC. In this work we provide two main results, we start by describing the first which is a round optimal MPC protocol that 1) is secure against malicious adversaries 2) tolerates arbitrary many parties, 3) is secure under standard polynomial time assumptions and 4) is circuit-scalable, i.e., has communication complexity proportional to the depth of the circuit and the length of the input and output of the circuit being evaluated.[7] We do so by extending the study of the connection between functional encryption combiners and secure MPC to the malicious setting. In summary, we prove the following theorem.

Theorem 1 (informal). *If there exists an ℓ-round MPC protocol Π[8] that is secure against malicious adversaries and a succinct FE combiner, then there exists an ℓ-round MPC protocol Π' that is secure against malicious adversaries whose communication efficiency depends only on the security parameter, the depth, the input length and the output length of the circuit being evaluated, and that makes black-box use of Π.*

Our compiler requires the input protocol Π to be k*-Signaling*. Informally, in a k-Signaling MPC protocol each party has a private and a public input. The private inputs of the honest parties are protected in the standard way, whereas the public input can be chosen (and signaled to the honest parties) by the adversary in the round $k-1$ of Π (this implies that the first $k-1$ rounds of Π are independent from the inputs).

We further show how to turn any MPC protocol that does not require the input to compute its first $k-1$ rounds into a k-Signaling MPC protocol (more details on this new notion are provided in the next section). This notion of k-Signaling MPC might be of independent interest. Finally, we observe that the 4-round protocols proposed in [BGJ$^+$18, CCG$^+$19] can be turned into k-Signaling MPC protocols, which in turn implies that we can obtain a communication efficient round optimal MPC protocol from the LWE assumption in combination with any of the assumptions DDH, QR, N$^{\text{th}}$ Residuosity, or existence of malicious-secure OT. This allows us to prove the following corollary.

Corollary 1 (informal). *If the LWE assumption holds and any of the assumptions DDH, QR, Nth Residuosity hold, or malicious-secure OT exists, then there exists a round optimal MPC protocol that is secure against malicious adversaries whose communication efficiency depends only on the security parameter, the depth, the input length and the output length of the circuit being evaluated.*

From MFHE to circuit-independent MPC. For the second contribution we show how to combine an MPC protocol with a perfect correct, compact MFHE scheme to obtain a *circuit-independent* MPC protocol. The notion of MFHE extends the notion of Fully-Homomorphic Encryption (FHE) to the multi-party setting by allowing each party to generate a public-secret key pair. All the ciphertexts generated using the public keys of the MFHE scheme can be homomorphically combined thus obtaining a single ciphertext, which can be decrypted only using all the secret keys.

[6] We recall that a semi-malicious adversary behaves like a semi-honest adversary with the exception that he decides the randomness and the input used to run the protocol.

[7] All our result are with respect to black-box simulation.

[8] We require an additional, non-standard property on the protocol Π that we discuss later.

The output of our compiler is a circuit-independent round optimal MPC protocol that tolerates $\min\{n_0, n_1\}$ parties where n_0 and n_1 is the number of parties tollerated by the input MPC protocol and the MFHE scheme respectively.

Our second contribution can be summarized as follows.

Theorem 2 (informal). *If there exists an ℓ-round MPC protocol Π*[9] *that is secure against malicious adversaries which tolerates n_0 number of parties and a perfect correct, compact MFHE scheme that tolerates n_1 number of parties, then there exists an ℓ-round MPC protocol Π' that is secure against malicious adversaries whose communication efficiency depends (polynomially) only on the security parameter, the input length and the output length of the circuit being evaluated, and that makes black-box use of Π and tolerates* $\min\{n_0, n_1\}$ *number of parties.*

Additionally, it is possible to improve the above result obtaining a protocol whose communication complexity depends only linearly to the input length (an polynomially to the output length and the security parameter), by relying on pseudorandom generators (PRG). Hence, we obtain an MPC protocol that is optimal in terms of round and communication complexity for all the functions whose input-size is bigger than the output-size (e.g, boolean functions).

Given that a MFHE scheme for constant number of parties can be instantiated from LWE and that a scheme for arbitrary many parties can be instantiated from Ring-LWE and DSPR [LTV12] we obtain the following additional corollary.

Corollary 2 (informal). *If the LWE (resp. Ring LWE and DSPR) assumption holds and any of the assumptions DDH, QR, N^{th} Residuosity hold, or malicious-secure OT exists, then there exists a round optimal circuit-independent MPC protocol for a constant (arbitrarily) number of parties that is secure against malicious adversaries.*

For completeness we have included a comprehensive comparison of our results with existing round-optimal MPC protocols proven secure in the plain-model, under standard polynomial-time complexity assumptions in Table 1.

2 Technical overview

In this section we give an overview of our two compilers.

From FE combiners to circuit-scalable MPC. Our starting point is the compiler proposed in [ABJ$^+$19]. The main building blocks of this compiler are an ℓ-round semi-honest secure MPC protocol and a succinct decomposable FE combiner. The property of decomposability requires the functional key for f to be of the form $(\mathsf{sk}_1^f, \ldots, \mathsf{sk}_n^f)$, and the master secret key needs to be $(\mathsf{msk}_1, \ldots, \mathsf{msk}_n)$, where sk_i and msk_i are the secret key and master secret key produced by the i-th FE candidate respectively.

The construction proposed in [ABJ$^+$19] is very intuitive, and roughly works as follows. The MPC protocol computes the function g which takes n inputs, one for each party P_i with $i \in [n]$. The input of each party consists of a master secret key msk_i, a value x_i and a randomness r_i. The function g uses the n master secret keys to compute an encryption of $x_1, \ldots, x_n$ using the randomness $r_1, \ldots, r_n$. Let x_i be the input of the party P_i with $i \in [n]$. Each party P_i samples a master secret key msk_i for the FE combiner, a random string r_i and runs the MPC protocol Π using $(\mathsf{msk}_i, x_i, r_i)$ as an input. In parallel, P_i computes the secret key sk_i^f and sends it to all the parties (we recall that sk_i^f can be computed by party P_i due to the decomposability property of the FE combiner). Let ct be the output of Π received by P_i, and let $(\mathsf{sk}_1^f, \ldots, \mathsf{sk}_{i-1}^f, \mathsf{sk}_{i+1}^f, \ldots, \mathsf{sk}_n^f)$ be the keys received from all the other parties, then P_i runs the decryption algorithm of the

[9] Also in this case we require Π to be k-signaling.

	Communication Complexity	Assumptions	Adversarial Model	Rounds
[ABJ+19, QWW18]	$\mathrm{poly}(\lambda, n, d, L_{\mathsf{in}}, L_{\mathsf{out}})$	LWE	Semi-honest	2
[BL18, GS18]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	Semi-honest OT	Semi-honest	2
[DHRW16]	$\mathrm{poly}(\lambda, n, d, L_{\mathsf{in}}, L_{\mathsf{out}})$	piO and lossy encryption	Semi-honest	2
[GS17]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	Bilinear Maps	Semi-honest	2
[HHPV18]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	QR	Malicious	4
[BGJ+18]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	DDH/QR/ N$^{\mathrm{th}}$ Residuosity	Malicious	4
[CCG+19]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	Malicious 4-round OT	Malicious	4
[AJJM20]	$\mathrm{poly}(\lambda, n, L_{\mathsf{in}}, L_{\mathsf{out}})$	Ring LWE and DSPR and 2-round OT	Semi-malicious	2
This work	$\mathrm{poly}(\lambda, n, d, L_{\mathsf{in}}, L_{\mathsf{out}})$	LWE and malicious 4-round OT	Malicious	4
This work*	$\mathrm{poly}(\lambda, n, L_{\mathsf{in}}, L_{\mathsf{out}})$	LWE and malicious 4-round OT	Malicious	4
This work	$\mathrm{poly}(\lambda, n, L_{\mathsf{in}}, L_{\mathsf{out}})$	Ring LWE and DSPR and malicious 4-round OT	Malicious	4

Table 1: Communication complexity of two-round semi-honest secure and four-round maliciously secure n-party protocols in the all-but-one corruption model, with black-box simulation, based on polynomial-time assumptions. We denote by $|f|$ and d the size and depth of the circuit representing the MPC functionality f, respectively. L_{in} and L_{out} denote, respectively, the input and output lengths of the circuit and piO stands for probabilistic indistinguishability obfuscation. We recall that we can replace 4-round maliciously secure OT with either DDH, QR, N$^{\mathrm{th}}$ Residuosity. *Constant number of parties only.

FE combiner on input $(\mathsf{sk}_1^f, \ldots, \mathsf{sk}_n^f)$ and ct thus obtaining $f(x_1, \ldots, x_n)$. Given that the MPC protocol computes a function g whose complexity is $\mathrm{poly}(\lambda, d, L_{\mathsf{in}})$ and the size of each one of the secret keys sent on the channel is $\mathrm{poly}(\lambda, d, L_{\mathsf{out}})$ the final protocol has communication complexity $\mathrm{poly}(\lambda, n, d, L_{\mathsf{in}}, L_{\mathsf{out}})$, where λ is the security parameter, d is the depth of f, L_{in} is the length of the input of f and L_{out} is the output length of f (we recall that this is due to the succinctness of the FE combiner). Starting from the above approach, we now show how to obtain a circuit-scalable MPC protocol in the case of malicious adversaries (other than semi-honest) in the plain model. As a first approach one can try to simply replace the semi-honest MPC protocol with a maliciously secure one. Unfortunately, this does not work as a corrupted party $P_j^\star$ might create an ill formed master secret key msk_j (i.e., msk_j is not generated accordingly to the setup procedure of the j-th FE candidate) and sample r_j according to an arbitrary strategy. However, we note that the second problem is straightforward to solve as we can modify the function g evaluated by the MPC protocol Π in such a way that it uses the randomness $r_1 \oplus \cdots \oplus r_n$ to compute the encryption ct (we note that in this case each party needs to sample a longer r_i compared to the semi-honest protocol described earlier).

To solve the first problem, we follow a similar approach. Each party P_i inputs an additional random value r_i^{Setup} to the MPC protocol and the function g is modified such that it generates the master secret keys using the randomness $R = r_1^{\mathsf{Setup}} \oplus \cdots \oplus r_n^{\mathsf{Setup}}$ and outputs to the party

P_i the ciphertext ct.[10] Unfortunately, this approach is not round preserving, as the knowledge of the master secret key msk_i, which becomes available only in the end of the execution of $\mathit{\Pi}$, is required to generate the secret key sk_i^f. Hence, if $\mathit{\Pi}$ requires ℓ-rounds, our final protocol would consist of $\ell + 1$ rounds as each party P_i needs to send its functional secret key sk_i^f in the $(\ell + 1)$-th round. Besides this, the described protocol is also still insecure, since a corrupted party $P_j^\star$ might generate an ill formed secret key sk_j^f, that could decrypt ct incorrectly, yielding an incorrect output for the honest parties. However, we can prove that this protocol protects the inputs of the honest parties. That is, it is secure under a notion called *privacy with knowledge of outputs (PKO)* [IKP10, PC12]. We now describe how to modify the above protocol in such a way that the round complexity is kept down to ℓ, while achieving privacy with knowledge of outputs. In the next step, we show how to promote any protocol that realizes any functionality with privacy with knowledge of outputs to a secure MPC protocol in a round preserving (and communication efficient) way.

Round preserving construction: privacy with knowledge of outputs. For simplicity, we describe our protocol considering only two parties P_0 and P_1 and consider as a building block an MPC protocol $\mathit{\Pi}$ which consists of $(\ell = 4)$-rounds only (which is optimal). The protocol then can be trivially extended to the case of n-parties and an arbitrary $\ell \geq 4$ as we show in the technical part of the paper.

For our construction we need the first two rounds of $\mathit{\Pi}$ to be independent of the inputs (i.e., the input is required only to compute the last two rounds in our simplified example). Assuming that the parties have access to a simultaneous broadcast channel where every party can simultaneously broadcast a message to all other parties, our compiler works as follows (we refer to Fig. 1 for a pictorial representation). Each party P_i commits to two random strings in the first round $c_i^0 := \mathsf{com}(r_i^0; \rho_i^0)$ and $c_i^1 := \mathsf{com}(r_i^1; \rho_i^1)$ and sends, in the second round, r_{1-i}^i to P_{1-i}.[11] Then P_i uses the randomness $R_i := r_0^i \oplus r_1^i$ to generate a master secret key msk_i, and uses it to compute the secret key sk_i^f which it sends in the fourth round. In parallel, P_0 and P_1 execute the MPC protocol $\mathit{\Pi}$ that evaluates the function g'. The function g' takes the inputs of each party, where the input corresponding to party P_i (for each $i \in \{0, 1\}$) is of the form $(x_i, (r_i^0; \rho_i^0, r_i^1; \rho_i^1, r_{1-i}^i, r_i,), (c_1^0, c_1^1, c_2^0, c_2^1))$. In more detail, the input of each party P_i corresponds to its actual input x_i, all the commitments generated (by P_0 and P_1) in the first round, the message r_{1-i}^i received in the second round from P_{1-i} and the randomness used to generate the commitments c_i^0, c_i^1. The function g' checks that 1) the commitments $(c_1^0, c_1^1, c_2^0, c_2^1)$ (that are part of the inputs of the two parties) are the same, 2) the value r_i^{1-i} sent in the second round by the party P_i is committed in c_i^{1-i} for each $i \in \{0, 1\}$ and 3) the randomness used to generate the commitments is correct. If all these checks are successful then g' outputs a ciphertext $\mathsf{ct} = \mathsf{Enc}((\mathsf{msk}_i)_{i\in\{0,1\}}, (x_0, x_1); r_0 \oplus r_1)$ for the FE combiner computed using the randomness $r_0 \oplus r_1$. Upon receiving the output of g' (evaluated by $\mathit{\Pi}$), P_i computes the output running the decryption algorithm of the FE combiner. Using this approach we guarantee that: 1) the ciphertext ct is computed honestly using honestly generated master secret keys and randomnesses, 2) each party can compute its own master secret key already in the third round so that a functional key can be generated and output in the last round and 3) the value r_{1-i}^i that P_i receives in the second round corresponds to the value used in the commitment c_{1-i}^i (hence, the master secret key that P_i obtains as part of the output of $\mathit{\Pi}$ is consistent with the master secret key he has created *outside* of $\mathit{\Pi}$). Unfortunately, we can only prove that the above protocol preserves the privacy of the inputs of the honest parties, but the output computed by

[10] R is parsed as n strings and each of the string is used to generate a different master secret key.

[11] Note that only the committed message is sent, not the randomness ρ_i^{1-i}.

the honest parties might still be incorrect. This is due to the fact that a corrupted party can generate an ill formed secret key sk_i^f and send it to the honest parties.

Full security. One way to solve this issue would be attaching a zero-knowledge proof to each secret key sk_i^f proving that sk_i^f is generated correctly (and accordingly to the master secret key generated by Π). It is easy to see that this would nullify our effort to construct a communication efficient protocol as the size of the proof would depend on the complexity of the key-generation algorithm of the FE combiner (that might depend on the size of the circuit f and not only on its depth). We might need a succinct 4-round delayed-input zero-knowledge argument system that can be securely composed in parallel with any MPC protocol.[12] Unfortunately, we are not aware of any protocols that enjoy such properties relying on standard assumptions. Luckily, the work of Ishai et al. [IKP10] proposes a compiler to promote a protocol that is secure under the notion of privacy with knowledge of outputs to a fully secure protocol (in the standard simulation based sense). The compiler of [IKP10] relies on unforgeable signatures only (which in turn rely on OWFs), and at a high level works as follows.

Each party generates a signing key and a verification key $(\mathsf{vk}_i, \mathsf{sk}_i)$ for a signature scheme. Then P_i runs Π' on input $(\mathsf{vk}_i, \mathsf{sk}_i, x_i)$ where x_i represents the input of P_i. The output of the MPC protocol consists of 1) the output of the function that the parties want to compute, denoted as y, and 2) n signatures of y, one for each signing key. In parallel with Π', each party sends its own verification key in the first round. A party P_i accepts y as a valid output if and only if all the signatures output by Π' verify accordingly to the verification keys sent by all the parties in the first round. For sake of completeness, in our work we show that the above compiler preserves the round and the communication complexity of the input protocol Π'. We note that this protocol has the limitation that only functionalities with a single output can be computed (i.e., all the parties get the same output). Indeed, to prove that the protocol is secure with unanimous abort[13] it becomes crucial that every party sees the same y (and the same signatures).

To achieve any functionality from a single-output functionality, we consider the following protocol Π''' (a similar approach has been used in [LP09, AJW11]). In Π''', each party P_i runs Π'' on its actual input and a secret key for a symmetric-key encryption scheme. The output of Π'' then contains a sequence of ciphertexts, where the i-th ciphertext contains the i-th output of the functionality encrypted using the secret key chosen by the party P_i. Also in this case, we will formally prove that the compiler preserves the round and the communication complexity of the input protocol (Π'' in this case).

A note on MPC. We recall that for the construction of Π' (the protocol that provides privacy but not correctness) we require the existence of an MPC protocol Π that uses the input of the parties only to compute the last two rounds. This property is achieved by many existing MPC protocols (e.g., [BGJ$^+$18, BL18, CCG$^+$19]). However, we note that to rely on the security of an MPC protocol we need the input of the honest parties to be specified before the real (ideal) world experiment starts. Therefore, the honest parties cannot choose an input that depends on (for example) the first two messages of the protocol. More formally, the input of the honest party cannot even be partially decided by the adversary. Contrary, in our construction we do need that part of the input of the honest parties is adversarially decided. Indeed, we recall that the input that each honest party provides to Π consists of its input, the randomness used to generate some commitments and *all* the commitments that it has seen (even those generated by the adversary). These commitments might be generated adaptively on the commitments received from the honest party and on the first message of Π (since the adversary is rushing).

[12] The delayed-input property guarantees that the zero-knowledge and the soundness property holds even in the case where the adversary decides the statement in the last round.

[13] The standard definition of secure MPC requires either all the honest parties to accept or to abort.

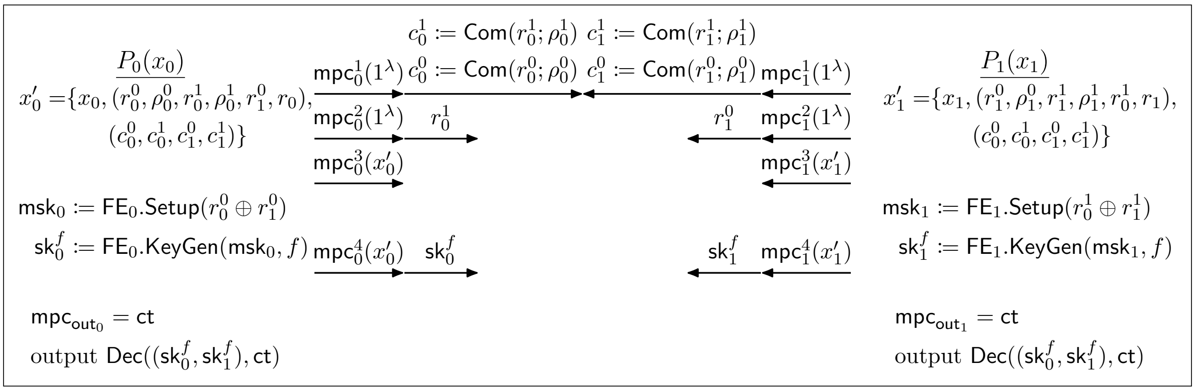
$c_0^1 := \mathsf{Com}(r_0^1; \rho_0^1)$
$c_1^1 := \mathsf{Com}(r_1^1; \rho_1^1)$
$P_0(x_0)$
$x_0' = \{x_0, (r_0^0, \rho_0^0, r_0^1, \rho_0^1, r_1^0, r_0), (c_0^0, c_0^1, c_1^0, c_1^1)\}$
$\mathsf{mpc}_0^1(1^\lambda)$
$c_0^0 := \mathsf{Com}(r_0^0; \rho_0^0)$
$c_1^0 := \mathsf{Com}(r_1^0; \rho_1^0)$
$\mathsf{mpc}_1^1(1^\lambda)$
$P_1(x_1)$
$x_1' = \{x_1, (r_1^0, \rho_1^0, r_1^1, \rho_1^1, r_0^1, r_1), (c_0^0, c_0^1, c_1^0, c_1^1)\}$
$\mathsf{mpc}_0^2(1^\lambda)$
r_0^1
r_1^0
$\mathsf{mpc}_1^2(1^\lambda)$
$\mathsf{mpc}_0^3(x_0')$
$\mathsf{mpc}_1^3(x_1')$
$\mathsf{msk}_0 := \mathsf{FE}_0.\mathsf{Setup}(r_0^0 \oplus r_1^0)$
$\mathsf{msk}_1 := \mathsf{FE}_1.\mathsf{Setup}(r_0^1 \oplus r_1^1)$
$\mathsf{sk}_0^f := \mathsf{FE}_0.\mathsf{KeyGen}(\mathsf{msk}_0, f)$
$\mathsf{mpc}_0^4(x_0')$
sk_0^f
sk_1^f
$\mathsf{mpc}_1^4(x_1')$
$\mathsf{sk}_1^f := \mathsf{FE}_1.\mathsf{KeyGen}(\mathsf{msk}_1, f)$
$\mathsf{mpc}_{\mathsf{out}_0} = \mathsf{ct}$
output $\mathsf{Dec}((\mathsf{sk}_0^f, \mathsf{sk}_1^f), \mathsf{ct})$
$\mathsf{mpc}_{\mathsf{out}_1} = \mathsf{ct}$
output $\mathsf{Dec}((\mathsf{sk}_0^f, \mathsf{sk}_1^f), \mathsf{ct})$

Fig. 1: FE_i, with $i \in \{0,1\}$, denotes a functional encryption candidate. The master secret key for the combiner corresponds to the master secret keys of FE_0 and FE_1. A secret key for the combiner required to evaluate the function f is generated by combining a secret key for FE_0 (sk_0^f) and a secret key for (sk_1^f). Dec denotes the decryption algorithm of the combiner which takes as input a combined secret key for the function f and a ciphertext ct generated accordingly to a combined master secret key represented by $(\mathsf{msk}_0, \mathsf{msk}_1)$. mpc_i^k, with $i \in \{0,1\}$ and $k \in [4]$, represents the k-th message of the MPC protocol Π computed by P_i. The protocol Π evaluates a function $g'(x_0', x_1')$ where $x_i' = \{x_i, (r_i^0, \rho_i^0, r_i^1, \rho_i^1, r_{1-i}^i, r_i), (c_0^0, c_0^1, c_1^0, c_1^1)\}$ with $i \in \{0,1\}$. The function g checks if the commitments that are part of the two inputs x_0', x_1' are the same and if c_i^b has been computed accordingly to the message r_i^b and the randomness ρ_i^b for each $i, b \in \{0,1\}$. If the check is successful, then g computes two master secret keys msk_0 and msk_1 using respectively the randomnesses $r_0^1 \oplus r_1^1$ and $r_0^0 \oplus r_1^0$, and computes an encryption ct of $x_0||x_1$ for the FE combiner using those master secret keys and the randomness $r_0 \oplus r_1$. The output of Π for P_i consists of $\mathsf{mpc}_{\mathsf{out}_i} = \mathsf{ct}$.

However, we observe that even if P_i needs to provide all the commitments it has received as part of its input to Π, we do not care about protecting the privacy of this part of P_i's input, we just want to achieve a correct evaluation of Π. We show how to construct an MPC protocol where the input of each party consists of two parts, a private part x and a public part w. The private part x is protected in the standard simulation based security sense, whereas the public part w can be determined adversarially, and does not need to be known before the experiment starts (either the ideal or the real world experiment). We refer to MPC protocols that are secure in this setting as *k-Signaling MPC* protocols (as part of the input is *signaled* in the $(k-1)$-th round by the adversary). We believe that this notion becomes fundamental in contexts where an MPC protocol is used as a building block, and it is crucial to keep the round complexity low by starting computing the messages of an MPC protocol before its inputs are fully specified. The same notion is required also to construct our second compiler.

From MFHE to circuit-independent MPC. To obtain circuit-independent MPC, we rely on a multi-key fully-homomorphic encryption scheme (MFHE) in combination with a non-communication-efficient MPC protocol Π. A MFHE scheme consists of four algorithms, a setup algorithm Setup that allows for the generation of public-secret key pair, an encryption algorithm Enc that takes as input a public key and a message and outputs a cipthertext, an evaluation algorithm Eval that takes as input a list of public keys PK, a set of cipthertext CT (generated using the list of public keys PK) and a function f, and outputs a ciphertexts ct that contains the evaluation of f on input the messages encrypted in the list CT. A decryption algorithm Dec that on input all the secret keys, associated with the public keys of PK, and the cipthertext ct outputs the decryption of ct. Additionally, we require the MFHE scheme to be *compact*, in more detail, we require the size of the keys, the ciphertexts and the description of the algorithms Enc and Dec to dependent only on the input-output size of f. For sake of simplicity, we describe our

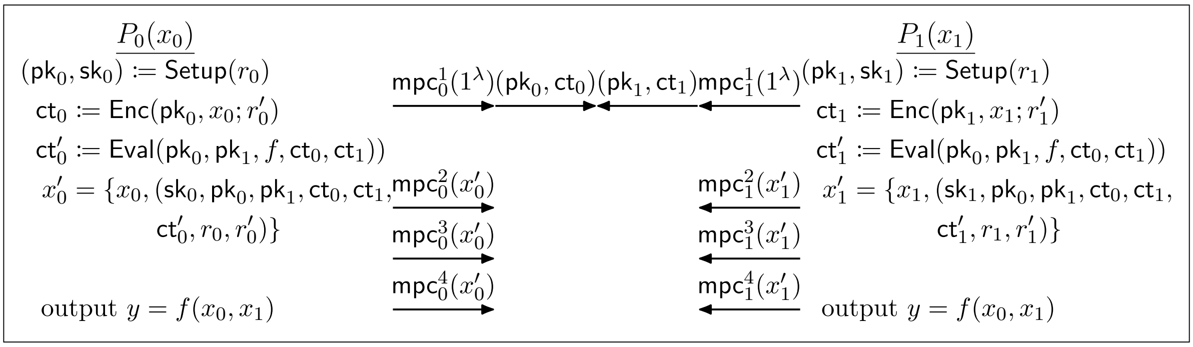
$P_0(x_0)$
$(\mathsf{pk}_0, \mathsf{sk}_0) := \mathsf{Setup}(r_0)$
$\mathsf{ct}_0 := \mathsf{Enc}(\mathsf{pk}_0, x_0; r'_0)$
$\mathsf{ct}'_0 := \mathsf{Eval}(\mathsf{pk}_0, \mathsf{pk}_1, f, \mathsf{ct}_0, \mathsf{ct}_1))$
$x'_0 = \{x_0, (\mathsf{sk}_0, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}'_0, r_0, r'_0)\}$
output $y = f(x_0, x_1)$
$\mathsf{mpc}^1_0(1^\lambda)(\mathsf{pk}_0, \mathsf{ct}_0)$
$(\mathsf{pk}_1, \mathsf{ct}_1)\mathsf{mpc}^1_1(1^\lambda)$
$\mathsf{mpc}^2_0(x'_0)$
$\mathsf{mpc}^3_0(x'_0)$
$\mathsf{mpc}^4_0(x'_0)$
$\mathsf{mpc}^2_1(x'_1)$
$\mathsf{mpc}^3_1(x'_1)$
$\mathsf{mpc}^4_1(x'_1)$
$P_1(x_1)$
$(\mathsf{pk}_1, \mathsf{sk}_1) := \mathsf{Setup}(r_1)$
$\mathsf{ct}_1 := \mathsf{Enc}(\mathsf{pk}_1, x_1; r'_1)$
$\mathsf{ct}'_1 := \mathsf{Eval}(\mathsf{pk}_0, \mathsf{pk}_1, f, \mathsf{ct}_0, \mathsf{ct}_1))$
$x'_1 = \{x_1, (\mathsf{sk}_1, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}'_1, r_1, r'_1)\}$
output $y = f(x_0, x_1)$

Fig. 2: $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ represents a MFHE scheme. The MPC protocol checks that the cipthertexes ct_0 and ct_1 are in the domain of Enc and that both parties have input the same list of cipthertexes $\mathsf{ct}_0, \mathsf{ct}_1$. Then the MPC protocol decrypts ct'_0 and ct'_1 and if the decrypted values corresponds to the same value y then the protocol outputs y.

compiler only for the two party case and refer to Section 5 to the description of the protocol that tolerates arbitrary many parties.

We provide a pictorial description of our protocol in Fig. 2. At a high level, our compiler works as follows. Let x_i be the secret input of the party P_i with $i \in \{0, 1\}$. Each party P_i runs the setup algorithm using the randomness r_i thus obtaining a private-secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ and encrypts its input using Enc with some randomness r'_i, obtaining ct_i. Then P_i sends the public key together with its encrypted output and the first message of the MPC protocol Π to party P_{1-i}. Upon receiving pk_{1-i} and ct_{1-i} from P_{i-1}, P_i runs the evaluation algorithm on input $\mathsf{pk}_0, \mathsf{pk}_1, f, \mathsf{ct}_0, \mathsf{ct}_1$, obtaining ct'_i. At this point P_i keeps executing the protocol Π on input x_i which consists of the randomness used to generate the MFHE keys, the randomness used to generate ct_i, the list of all the ciphertexts (received and generated) $\mathsf{CT} = (\mathsf{ct}_0, \mathsf{ct}_1)$ and the evaluated ciphertext ct'_i. The function g computed by the MPC protocol Π does the following: 1) checks that both P_0 and P_i have input the same list of ciphertexts CT, 2) for each $i \in \{0, 1\}$ uses the randomness r_i and r'_i to check that pk_i and ct_i are in the domain of the setup and of the encryption algorithm respectively. If these checks are successful, then the function g decrypts ct'_0 and ct'_1 using the secret keys $(\mathsf{sk}_0, \mathsf{sk}_1)$ (which can be generated using the randomnesses r_0, r_1) thus obtaining y_0 and y_1. If $y_0 = y_1$ then g outputs y, otherwise it outputs $\perp$. In a nutshell, we use Π to check that all cipthertexts and public keys have been generated correctly and that all the parties have obtained an encryption of the same value by running the MFHE evaluation algorithm. The protocol that we have just described is circuit-independent since the size of the public keys and the cipthertexts depends only on the input-output size of f and the protocol Π evaluates a function g whose description size depends only on the input-output size of f and the description of the circuits for Enc and Dec.

The communication complexity of this protocol is $\mathrm{poly}(\lambda, n, L_{\mathsf{in}}, L_{\mathsf{out}})$, where L_{in} is the input-size and L_{out} is the output size of the function being evaluated. We can slightly modify the protocol above to get communication complexity $O(L_{\mathsf{in}}) + \mathrm{poly}(\lambda, n, L_{\mathsf{out}})$. To do that, we can rely on a folklore technique to reduce the size of the cipthertexts of the MFHE scheme relying on pseudorandom generators (PRGs). In more detail, instead of providing an encryption of the input x_i under the MFHE scheme, each party P_i encrypts a short seed s_i of a PRG PRG using the FHE scheme, i.e. $\mathsf{Enc}(\mathsf{pk}_i, s_i; r^s_i)$, and sends this encryption along with the value $w_i = \mathsf{PRG}(s_i) \oplus x_i$ to the other party. This size of the resulting message is then $O(L_{\mathsf{in}}) + \mathrm{poly}(\lambda)$. The party P_i, upon receiving $(\mathsf{Enc}(\mathsf{pk}_{1-i}, s_{1-i}; r^s_{1-i}), w_{1-i})$ computes $\mathsf{Enc}(\mathsf{pk}_{1-i}, \mathsf{PRG}(s_{1-i}))$, using homomorphic operations, $\mathsf{Enc}(\mathsf{pk}_{1-i}, w_{i-1})$ by encrypting w_{1-i} using pk_{1-i}, and then homomorphically XORs the resulting ciphertexts to receive $\mathsf{Enc}(\mathsf{pk}_{1-i}, x_{1-i})$. This ciphertext can now be used to run the evaluation algorithm and compute $\mathsf{Enc}(\mathsf{pk}_0, \mathsf{pk}_1, f(x_0, x_1))$. The parties now check that the ciphertexts (w_0, w_1) are well formed by running the MPC protocol, exactly as in the previous protocol.

We note that the above protocols require Π to be 2-signaling. Moreover, we need the MFHE scheme to achieve perfect correctness. Removing this additional assumption remains an interesting open problem.

3 Preliminaries

Before diving into the technical part, let us first introduce some needed terminology along with formal definitions of the concepts and primitives used in our construction and analysis. A reader familiar with the relevant literature can skip this part and refer to it later as needed.

Notation. We denote the security parameter with $\lambda \in \mathbb{N}$. A randomized algorithm $\mathcal{A}$ is running in *probabilistic polynomial time* (PPT) if there exists a polynomial $p(\cdot)$ such that for every input x the running time of $\mathcal{A}(x)$ is bounded by $p(|x|)$. We call a function $\mathrm{negl} : \mathbb{N} \to \mathbb{R}^+$ *negligible* if for every positive polynomial $p(\lambda)$ a $\lambda_0 \in \mathbb{N}$ exists, such that for all $\lambda > \lambda_0 : \epsilon(\lambda) < 1/p(\lambda)$. We denote by $[n]$ the set $\{1, \ldots, n\}$ for $n \in \mathbb{N}$. We use "$=$" to check equality of two different elements (i.e. $a = b$ then...) and "$:=$" as the assigning operator (e.g. to assign to a the value of b we write $a := b$). A randomized assignment is denoted with $a \leftarrow A$, where A is a randomized algorithm and the randomness used by A is not explicit. If the randomness is explicit we write $a := A(x; r)$ where x is the input and r is the randomness.

3.1 Functional Encryption

In this section, we recap the notion of (secret key) functional encryption (FE) [SW05, BSW11, O'N10].

Definition 1 (Functional Encryption). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a collection of circuit families (indexed by λ), where every $C \in \mathcal{C}_\lambda$ is a polynomial time circuit $C \colon \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$. A (secret-key) functional encryption scheme (FE) for the circuit family $\mathcal{C}_\lambda$ is a tuple of four algorithms* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as input a unary representation of the security parameter λ and generates a master secret key* msk*. It also outputs the randomness r that has been used to generate the master secret key.*

$\mathsf{KeyGen}(\mathsf{msk}, C)$**:** *Takes as input the master secret key* msk *and a circuit $C \in \mathcal{C}_\lambda$, and outputs a functional key* sk_C*.*

$\mathsf{Enc}(\mathsf{msk}, x)$**:** *Takes as input the master secret key* msk*, a message $x \in \mathcal{X}_\lambda$ to encrypt, and outputs a ciphertext* ct*.*

$\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct})$**:** *Is a deterministic algorithm that takes as input a functional key* sk_C *and a ciphertext* ct *and outputs a value $y \in \mathcal{Y}_\lambda$.*

A scheme FE *is (approximate) correct, if for all $\lambda \in \mathbb{N}$,* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$*, $C \in \mathcal{C}_\lambda$, $x \in \mathcal{X}_\lambda$, when* $\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$*, we have*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{Enc}(\mathsf{msk}, x)) = C(x)\right] \geq 1 - \mathrm{negl}(\lambda) \ ,$$

for a negligible function $\mathrm{negl}(\cdot)$*.*

In contrast to the standard definition of secret key functional encryption, where the setup algorithm just outputs a master secret key msk, we define our scheme in such a way that it also outputs the randomness r, that has been used to generate the master secret key. This has no effects on the security definition of the scheme since the master secret key msk and the randomness r both remain in the control of the challenger. We need to rely on this little modification later to enforce a party to generate a specific master secret key.

Now, we recall the definition of single key simulation security for a functional encryption scheme as stated in [ABJ+19].

Definition 2 (Single Key Simulation Security of FE). *Let* FE *be a functional encryption scheme,* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *a collection of circuit families indexed by* λ*. We define the experiments* $\mathrm{Real}^{\mathsf{DFEC}}$ *and* $\mathrm{Ideal}^{\mathsf{DFEC}}$ *in Fig. 3. A functional encryption scheme* FE *is single key simulation secure, if for any polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *exists a PPT simulator* $\mathcal{S}$ *and a negligible function* negl *such that:*

$$|\Pr[\mathrm{Real}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathrm{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S}) = 1]| \leq \mathrm{negl}(\lambda) \ .$$

$\mathbf{Real}^{\mathsf{FE}}(1^\lambda, \mathcal{A})$	$\mathbf{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$
$\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$	$\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$
$(C, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\lambda)$	$(C, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\lambda)$
$\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$	$\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$
$(x, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{sk}_C, \mathsf{st}_1)$	$(x, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{sk}_C, \mathsf{st}_1)$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, x)$	$\mathsf{ct} \leftarrow \mathcal{S}(\mathsf{msk}, C, C(x))$
$\alpha \leftarrow \mathcal{A}_3(\mathsf{ct}, \mathsf{sk}_C, \mathsf{st}_2)$	$\alpha \leftarrow \mathcal{A}_3(\mathsf{ct}, \mathsf{sk}_C, \mathsf{st}_2)$
Output: α	Output: α

Fig. 3: Single Key Simulation Security of FE

The succinctness definition provided in [ABJ+19] requires some restrictions on the circuit size of the encryption algorithm, as well as on the size of the functional key. In our work, we also require a bounded circuit size for the setup algorithm and we refer to this notion as strong succinctness.

Definition 3 (Strong Succinctness). *A functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *for a circuit class* $\mathcal{C}$ *containing circuits that take inputs of length* ℓ_{in}*, outputs strings of length* ℓ_{out} *bits and are of depth at most* d *is succinct if the following holds: For any circuit* $C \in \mathcal{C}$

- *For the size of the circuit* $\mathsf{Setup}(1^\lambda)$ *is* $\mathrm{poly}(\lambda, d, \ell_{\mathsf{in}})$ *for some polynomial* poly*.*
- *Let* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$*. The size of the circuit* $\mathsf{Enc}(\mathsf{msk}, \cdot)$ *is* $\mathrm{poly}(\lambda, d, \ell_{\mathsf{in}}, \ell_{\mathsf{out}})$ *for some polynomial* poly*.*
- *The functional key* $\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$ *is of the form* (C, aux) *where* $|\mathsf{aux}| \leq \mathrm{poly}(\lambda, d, \ell_{\mathsf{out}}, n)$ *for some polynomial* poly*.*

3.2 Decomposable Functional Encryption Combiner

After recapping the notion of functional encryption, we are ready to define a decomposable functional encryption combiner (DFEC) as introduced by Ananth et al. [ABJ+19].

Definition 4 (Decomposable Functional Encryption Combiner). *Let* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *be a collection of circuit families (indexed by* λ*), where every* $C \in \mathcal{C}_\lambda$ *is a polynomial time* *circuit* $C\colon \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$ *and let* $\{\mathsf{FE}_i\}_{i\in[n]}$ *be the description of n FE candidates. A decomposable functional encryption combiner (DFEC) for the circuit family* $\mathcal{C}_\lambda$ *is a tuple of five algorithms* $\mathsf{DFEC} = (\mathsf{Setup}, \mathsf{Partition}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*:*

$\mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$**:** *Takes as input a unary representation of the security parameter* λ *and the description of n FE candidates* $\{\mathsf{FE}_i\}_{i\in[n]}$ *and generates a master key* msk_i *for each FE candidate* $\mathsf{msk}_i \leftarrow \mathsf{FE.Setup}_i(1^\lambda)$ *and outputs* $\mathsf{msk} := \{\mathsf{msk}_i\}_{i\in[n]}$.

$\mathsf{Partition}(n, C)$**:** *Takes as input the number of parties n and a circuit C and outputs* $(C_1, \dots, C_n)$*, where each* C_i *is a circuit of depth polynomial in the depth of* C.

$\mathsf{KeyGen}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, (C_1, \dots, C_n))$**:** *Takes as input the master secret key* msk*, the description of n FE candidates* $\{\mathsf{FE}_i\}_{i\in[n]}$*, a partitioned circuit* $(C_1, \dots, C_n)$*, generates a functional key* sk_{C_i} *for each FE candidate* $\mathsf{sk}_{C_i} \leftarrow \mathsf{FE.KeyGen}_i(\mathsf{msk}_i, C_i)$ *and outputs* $\mathsf{sk}_C := \{\mathsf{sk}_{C_i}\}_{i\in[n]}$.

$\mathsf{Enc}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, x)$**:** *Takes as input the master secret key* msk*, the description of n FE candidates* $\{\mathsf{FE}_i\}_{i\in[n]}$*, a message* $x \in \mathcal{X}_\lambda$ *to encrypt, and outputs a ciphertext* ct.

$\mathsf{Dec}(\mathsf{sk}_C, \{\mathsf{FE}_i\}_{i\in[n]}, \mathsf{ct})$**:** *Is a deterministic algorithm that takes as input a functional key* sk_C*, the description of n FE candidates* $\{\mathsf{FE}_i\}_{i\in[n]}$ *and a ciphertext* ct *and outputs a value* $y \in \mathcal{Y}_\lambda$.

A scheme DFEC *is (approximate) correct, if for all* $\lambda \in \mathbb{N}$, $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$, $C \in \mathcal{C}_\lambda$, $x \in \mathcal{X}_\lambda$*, when* $\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$*, we have*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{Enc}(\mathsf{msk}, x)) = C(x)\right] \geq 1 - \mathrm{negl}(\lambda) \enspace ,$$

for a negligible function $\mathrm{negl}(\cdot)$.

The notion of decomposability is natural if a functional encryption combiner is considered in the context of multiparty computation. Since each party in an MPC protocol generates messages that are sent to the other parties it is intuitive to also allow each party to generate a functional key corresponding to their FE candidate that can be sent to everyone. The functional key of the FE combiner used in the decryption procedure together with the messages sent by each party contains of the functional keys of all the participating parties.

To ensure that all the algorithms of the functional encryption combiner are still polynomial in the security parameter λ and the number of parties n, we need to introduce the notion of polynomial slowdown.

Definition 5 (Polynomial Slowdown [ABJ+19]). *A decomposable functional encryption combiner* $\mathsf{DFEC} = (\mathsf{Setup}, \mathsf{Partition}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *satisifes polynomial slowdown, if the running time of all its algorithms are at most* $\mathrm{poly}(\lambda, n)$*, where n is the number of FE candidates that are being combined.*

The definition of single key simulation security of a functional encryption combiner should capture the case that if at least one of the FE candidates is secure, then the combiner is also secure. In the case of decomposability we give the adversary even more power by letting it choose a set I of all the corrupted candidates, which contains all but one party.

Definition 6 (Single Key Simulation Security of DFEC [ABJ+19]). *Let* DFEC *be a decomposable functional encryption combiner,* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *a collection of circuit families indexed by* λ *and* $\{\mathsf{FE}_i\}_{i\in[n]}$ *n FE candidates of which at least one is guaranteed to be secure. We define the experiments* $\mathrm{Real}^{\mathsf{DFEC}}$ *and* $\mathrm{Ideal}^{\mathsf{DFEC}}$ *in Fig. 4. A decomposable functional encryption combiner* DFEC *is single key simulation secure, if for any polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *exists a PPT simulator* $\mathcal{S}$ *and a negligible function* negl *such that:*

$$|\Pr[\mathrm{Real}^{\mathsf{DFEC}}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C, \mathcal{A}) = 1] - \Pr[\mathrm{Ideal}^{\mathsf{DFEC}}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C, \mathcal{A}, \mathcal{S}) = 1]| \leq \mathrm{negl}(\lambda) \enspace .$$

$\mathbf{Real}^{\mathsf{DFEC}}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C, \mathcal{A})$	$\mathbf{Ideal}^{\mathsf{DFEC}}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C, \mathcal{A}, \mathcal{S})$
$\mathsf{msk} := \{\mathsf{msk}_i\}_{i\in[n]} \leftarrow \mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$	$\mathsf{msk} := \{\mathsf{msk}_i\}_{i\in[n]} \leftarrow \mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$
$(C_1, \ldots, C_n) = \mathsf{Partition}(n, C)$	$(C_1, \ldots, C_n) = \mathsf{Partition}(n, C)$
$\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, (C_1, \ldots, C_n))$	$\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, (C_1, \ldots, C_n))$
$(I, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C)$, where $I \subset [n]$ with $\lvert I \rvert = n - 1$.	$(I, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C)$, where $I \subset [n]$ with $\lvert I \rvert = n - 1$.
$(x, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\{\mathsf{msk}_i\}_{i\in I}, \mathsf{sk}_C, \mathsf{st}_1)$	$(x, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\{\mathsf{msk}_i\}_{i\in I}, \mathsf{sk}_C, \mathsf{st}_1)$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, x)$	$\mathsf{ct} \leftarrow \mathcal{S}(\mathsf{msk}, C, C(x))$
$\alpha \leftarrow \mathcal{A}_3(\mathsf{ct}, \mathsf{sk}_C, \mathsf{st}_2)$	$\alpha \leftarrow \mathcal{A}_3(\mathsf{ct}, \mathsf{sk}_C, \mathsf{st}_2)$
Output: α	Output: α

Fig. 4: Single Key Simulation Security of DFEC

As in the case of an FE candidate, we also need to define succinctness for an FE combiner. We, again, adapt the notion of Ananth et al. [ABJ+19] here and present our notion of strong succinctness.

Definition 7 (Strong Succinctness). *A decomposable FE combiner* $\mathsf{DFEC} = (\mathsf{Setup}, \mathsf{Partition}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *for a circuit class* $\mathcal{C}$ *containing circuits that take inputs of length* ℓ_{in}*, outputs strings of length* ℓ_{out} *bits and are of depth at most* d *is succinct if for every set of succinct FE candidates* $\{\mathsf{FE}_i\}_{i\in[n]}$*, the following holds:*

- *For the size of the circuit* $\mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$ *it holds that* $\mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}) \leq \mathrm{poly}(\lambda, n, d, \ell_{\mathsf{in}})$*.*
- *Let* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$*. For the circuit* $\mathsf{Enc}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, \cdot)$ *it holds that* $\mathsf{Enc}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, \cdot) \leq \mathrm{poly}(\lambda, d, \ell_{\mathsf{in}}, \ell_{\mathsf{out}}, n)$ *for some polynomial* poly*.*
- *The functional key* $\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, (C_1, \ldots, C_n))$*, with* $(C_1, \ldots, C_n) = \mathsf{Partition}(n, C)$*, is of the form* (C, aux) *where* $\lvert\mathsf{aux}\rvert \leq \mathrm{poly}(\lambda, d, \ell_{\mathsf{out}}, n)$ *for some polynomial* poly*.*

3.3 Multi Key Fully Homomorphic Encryption

In this section, we recap the definition definitions of multi key fully homomorphic encryption (MFHE) as introduced by López-Alt, Tromer, and Vaikuntanathan [LTV12].

Definition 8 (Multi Key Fully Homomorphic Encryption). *Let* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *be a collection of circuit families (indexed by* λ*), where every* $C \in \mathcal{C}_\lambda$ *is a polynomial time circuit* $C\colon \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$ *and* n *the number of participating parties. A multi key fully homomorphic encryption (MFHE) for the circuit family* $\mathcal{C}_\lambda$ *is a tuple of four algorithms* $\mathsf{MFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$*:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as input a unary representation of the security parameter* λ *and generates a public key* pk *and a secret key* sk*.*

$\mathsf{Enc}(\mathsf{pk}, x)$**:** *Takes as input a public key* pk *and a message* $x \in \mathcal{X}_\lambda$ *to encrypt, and outputs a ciphertext* ct*.*

$\mathsf{Eval}(C, (\mathsf{pk}_i, \mathsf{ct}_i)_{i\in[\ell]})$**:** *Takes as input a circuit* C*,* ℓ *different public keys* pk_i *and ciphertexts* ct_i *and outputs a ciphertext* ct*.*

$\mathsf{Dec}(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{ct})$**:** *Is a deterministic algorithm that takes as input* n *secret keys* $\{\mathsf{sk}_i\}_{i\in[n]}$ *and a ciphertext* ct *and outputs a value* y*.*

```
$\textbf{IND-CPA}_{\beta}^{\mathsf{MFHE}}(1^{\lambda}, \mathcal{A})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^{\lambda})$
$(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pk})$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x_{\beta})$
$\alpha \leftarrow \mathcal{A}_2(\mathsf{st}, \mathsf{ct})$
Output: $\alpha$
```

Fig. 5: The IND-CPA Game.

A scheme MFHE *is perfectly correct, if for all $\lambda \in \mathbb{N}$, $i \in [n]$, $\ell \leq n$, $r_i^{\mathsf{Setup}} \leftarrow \{0,1\}^{\lambda}, r_i^{\mathsf{Enc}} \leftarrow \{0,1\}^{\lambda}$, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Setup}(1^{\lambda}; r_i^{\mathsf{Setup}})$, $C \in \mathcal{C}_{\lambda}$, $x_i \in \mathcal{X}_{\lambda}$, we have*

$$\Pr\left[\mathsf{Dec}(\{\mathsf{sk}_i\}_{i\in[n]}, \mathsf{Eval}(C, (\mathsf{pk}_i, \mathsf{Enc}(\mathsf{pk}_i, x_i; r_i^{\mathsf{Enc}}))_{i\in[\ell]})) = C(x_1, \ldots, x_{\ell})\right] = 1.$$

For $n = 1$ multi key FHE is equivalent to FHE. In the introductory paper of López-Alt, Tromer, and Vaikuntanathan [LTV12], the setup algorithm also outputs an evaluation key together with the public and secret key. In our work we assume that the information of the evaluation key is contained in the public key.

Definition 9 (IND-CPA security of MFHE). *Let* $\mathsf{MFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *be a MFHE scheme. For $\beta \in \{0,1\}$, we define the experiment* $\text{IND-CPA}_{\beta}^{\mathsf{MFHE}}$ *in Fig. 5, where the advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is defined by*

$$\mathsf{Adv}_{\mathsf{MFHE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) = |\Pr[\text{IND-CPA}_0^{\mathsf{MFHE}}(\lambda, \mathcal{A}) = 1] - \Pr[\text{IND-CPA}_1^{\mathsf{MFHE}}(\lambda, \mathcal{A}) = 1]|.$$

A multi key fully homomorphic encryption scheme MFHE *is called secure, if for any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function* negl *such that:* $\mathsf{Adv}_{\mathsf{MFHE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) \leq \text{negl}(\lambda)$.

Besides the security of a multi key FHE scheme, we also need to define what it means for a multi key FHE scheme to be compact.

Definition 10 (Compactness). *A multi key FHE scheme* $\mathsf{MFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Enc}, \mathsf{Dec})$ *for a circuit class $\mathcal{C}$ and n participating parties is called compact, if $|\mathsf{ct}| \leq \text{poly}(\lambda, n)$, where* $\mathsf{ct} := \mathsf{Eval}(C, (\mathsf{pk}_i, \mathsf{ct}_i)_{i\in[\ell]})$ *with $\ell \leq n$ and with description of the circuits* $\mathsf{Setup}, \mathsf{Enc}$ *and* Dec *being polynomial in the security parameter λ.*

We note that this definition implies that public and secret key pairs are also independent from the size of the circuit.

3.4 Symmetric Encryption, Authentication and Commitments

In this section, we recall the definitions of symmetric encryption, message authentication codes, digital signatures, and commitments.

Definition 11 (Symmetric Encryption [GB96]). *A symmetric encryption scheme (SE) for the message space $\mathcal{M}$ is a tuple of three algorithms* $\mathsf{SE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$*:*

$\mathsf{Setup}(1^{\lambda})$**:** *Takes as input a unary representation of the security parameter λ, and outputs a key* k.

$\mathsf{Enc}(\mathsf{k}, m)$**:** *Takes as input the symmetric key* k*, a message* $m \in \mathcal{M}$ *to encrypt, and outputs a ciphertext* ct*.*

$\mathsf{Dec}(\mathsf{k}, \mathsf{ct})$**:** *Takes as input the symmetric key* k *and a ciphertext* ct *and outputs a message or* $\perp$ *if decryption fails.*

A scheme SE *is correct, if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{k} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $m \in \mathcal{M}$*, we have*

$$\Pr\left[\mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}, m)) = m\right] = 1 \ .$$

Security for a symmetric encryption scheme is defined in an indistinguishable manner.

Definition 12 (IND-CPA Security of SE). *Let* $\mathsf{SE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *be an SE scheme, for the message space* $\mathcal{M}$*. For* $\beta \in \{0,1\}$*, we define the experiment* $\text{IND-CPA}_\beta^{\mathsf{SE}}$ *in Fig. 6, where the encryption oracle* QEnc *outputs* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{k}, x_\beta)$ *on a query* (x_0, x_1)*. We define the advantage of an adversary* $\mathcal{A}$ *in the following way*

$$\mathsf{Adv}_{\mathsf{SE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) = |\Pr[\text{IND-CPA}_0^{\mathsf{SE}}(\lambda, \mathcal{A}) = 1] - \Pr[\text{IND-CPA}_1^{\mathsf{SE}}(\lambda) = 1]| \ .$$

A symmetric encryption scheme SE *is called IND-CPA secure, if for any PPT adversary* $\mathcal{A}$ *it holds that* $\mathsf{Adv}_{\mathsf{SE},\mathcal{A}}^{\text{IND-CPA}}(\lambda) \leq \text{negl}(\lambda)$*.*

```
**IND-CPA**$_\beta^{\mathsf{SE}}(1^\lambda, \mathcal{A})$
$\mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEnc}(\cdot,\cdot)}(1^\lambda)$
**Output:** $\alpha$
```

Fig. 6: IND-CPA Security Game for a symmetric encryption scheme SE.

In our protocol, it is sufficient to use a symmetric encryption scheme that fulfills *one-time security.* In this special case of IND-CPA security the encryption oracle can only be queried once. The one-time pad is a candidate scheme that fulfills this notion.

Definition 13 (Message Authentication Code [Gol04]). *A message authentication code (MAC) for the message space* $\mathcal{M}$ *is a tuple of three algorithms* $(\mathsf{Setup}, \mathsf{Auth}, \mathsf{Verify})$*:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as and input a unary representation of the security parameter* 1^λ*, and outputs a key* k*.*

$\mathsf{Auth}(\mathsf{k}, m)$**:** *Takes as input the key* k*, a message* $m \in \mathcal{M}$*, and outputs a tag* τ*.*

$\mathsf{Verify}(\mathsf{k}, m)$**:** *takes as input a key* k*, a message* m *and a tag* τ *and outputs either* 0 *or* 1*.*

A scheme MAC *is correct, if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{k} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $m \in \mathcal{M}$*, we have*

$$\Pr\left[\mathsf{Verify}(\mathsf{k}, m, \mathsf{Auth}(\mathsf{k}, m)) = 1\right] = 1 \ .$$

Definition 14 (Unforgeability of MAC). *Let* $\mathsf{MAC} = (\mathsf{Setup}, \mathsf{Auth}, \mathsf{Verify})$ *be a MAC, for the message space* $\mathcal{M}$*. We define the experiment* $\text{EUF-CMA}^{\mathsf{MAC}}$ *in Fig. 7 with* Q *being the set containing the queries of* $\mathcal{A}$ *to the authentication oracle* $\mathsf{Auth}(\mathsf{k}, \cdot)$*.*

A message authentication code MAC *is called existentially unforgeable under adaptive chosen-message attacks (EUF-CMA secure), if for any PPT adversary* $\mathcal{A}$ *it holds that* $\Pr[\text{EUF-CMA}^{\mathsf{MAC}}(\lambda, \mathcal{A}) = 1] \leq \text{negl}(\lambda)$*.*

```
$\mathbf{EUF\text{-}CMA}^{\mathsf{MAC}}(1^\lambda, \mathcal{A})$
$\mathsf{k} \leftarrow \mathsf{Setup}(1^\lambda)$
$(m^*, \tau^*) \leftarrow \mathcal{A}^{\mathsf{Auth}(\mathsf{k},\cdot)}(1^\lambda)$
Output: $\mathsf{Verify}(\mathsf{k}, m^*, \tau^*) = 1 \land m \notin Q$
```

Fig. 7: The Existentially Unforgeability Game for a message authentication code MAC.

We say that a MAC is *one-time* if the adversary can query its oracle only once.

Definition 15 (Digital Signature Scheme [Can03]). *A digital signature scheme (DS) for the message space $\mathcal{M}$ is a tuple of three algorithms* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$*:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as input a unary representation of the security parameter λ, and outputs a verification key* vk *and a signing key* sk*.*

$\mathsf{Sign}(\mathsf{sk}, m)$**:** *Takes as input the signing key* sk*, a message $m \in \mathcal{M}$ to, and outputs a signature σ.*

$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$**:** *Takes as input the verification key* vk*, a message $m \in \mathcal{X}$ and a signature σ and outputs either 0 or 1.*

A scheme DS *is correct, if for all $\lambda \in \mathbb{N}$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $m \in \mathcal{M}$, we have*

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1\right] = 1 \ .$$

A scheme DS *is consistent, if for any $m \in \mathcal{X}$, the probability that $\mathsf{Setup}(1^\lambda)$ generates $(\mathsf{vk}, \mathsf{sk})$ and $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$ generates two different outputs in two independent invocations is 0.*

Definition 16 (Unforgeability of DS). *Let* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *be a DS scheme, for the message space $\mathcal{M}$. We define the experiment* $\text{EUF-CMA}^{\mathsf{DS}}$ *in Fig. 8 with Q being the set containing the queries of $\mathcal{A}$ to the signing oracle* $\mathsf{Sign}(\mathsf{sk}, \cdot)$*.*

A digital signature scheme DS *is called existentially unforgeable under adaptive chosen-message attacks (EUF-CMA secure), if for any PPT adversary $\mathcal{A}$ it holds that* $\Pr[\text{EUF-CMA}^{\mathsf{DS}}(\lambda, \mathcal{A}) = 1] \leq \text{negl}(\lambda)$*.*

```
$\mathbf{EUF\text{-}CMA}^{\mathsf{DS}}(1^\lambda, \mathcal{A})$
$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{vk})$
Output: $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1 \land m \notin Q$
```

Fig. 8: The Existentially Unforgeability Game for a signature scheme DS.

We recap the definition of a commitment scheme as stated in [Lin10] as well the definition of (computational) hiding and binding.

Definition 17 (Commitment Scheme). *A commitment scheme (CS) is a PPT algorithm* Com *that takes as an input a unary representation of the security parameter 1^λ, a message m a random value r and outputs a commitment.*

The pair (m, r) is called the decommitment of c.

We recall that commitments are secure under parallel composition. We will use this fact in the security proof of our compiler using the functional encryption combiner.

Definition 18 (Hiding of CS). *Let* Com *be a CS scheme, then we define the experiment* $\mathrm{HIDE}_\beta^{\mathsf{Com}}$ *in Fig. 9 The advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *is defined in the following way:*

$$\mathsf{Adv}_{\mathsf{Com},\mathcal{A}}^{\mathrm{HIDE}}(\lambda) = |\Pr[\mathrm{HIDE}_0^{\mathsf{Com}}(\lambda, \mathcal{A}) = 1] - \Pr[\mathrm{HIDE}_1^{\mathsf{Com}}(\lambda) = 1]| \ .$$

A commitment scheme Com *is called computational hiding, if for any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *it holds that* $\mathsf{Adv}_{\mathsf{Com},\mathcal{A}}^{\mathrm{HIDE}}(\lambda) \leq \mathrm{negl}(\lambda)$ *and perfectly hiding if* $\mathsf{Adv}_{\mathsf{Com},\mathcal{A}}^{\mathrm{HIDE}}(\lambda) = 0$.

```
**HIDE**$_\beta^{\mathsf{Com}}(1^\lambda, \mathcal{A})$
$(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda)$
$r \leftarrow \{0,1\}^\lambda$
$c = \mathsf{Com}(m_\beta; r)$
$\alpha \leftarrow \mathcal{A}_2(\mathsf{st}, c)$
**Output:** $\alpha$
```

Fig. 9: Hiding Game for a commitment scheme CS.

Definition 19 (Binding of CS). *Let* Com *be a CS scheme, then we define the experiment* $\mathrm{BIND}^{\mathsf{Com}}$ *in Fig. 10.*

A commitment scheme Com *is called computational binding, if for any PPT adversary* $\mathcal{A}$ *it holds that* $\Pr[\mathrm{BIND}^{\mathsf{Com}}(\lambda, \mathcal{A}) = 1] \leq \mathrm{negl}(\lambda)$ *and perfectly binding if* $\Pr[\mathrm{BIND}^{\mathsf{Com}}(\lambda, \mathcal{A}) = 1] = 0$.

```
**BIND**$^{\mathsf{Com}}(1^\lambda, \mathcal{A})$
$(c, m, r, m', r') \leftarrow \mathcal{A}(1^\lambda)$
**Output:** 1 if $\mathsf{Com}(m; r) = \mathsf{Com}(m'; r')$
            and 0 otherwise
```

Fig. 10: Binding Game for a commitment scheme CS.

3.5 Secure Multiparty Computation

We provide the definition of MPC against malicious adversaries. Parts of this section have been taken verbatim from [Gol04].

A multi-party protocol is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality. The security of a protocol is defined with respect to a functionality f. In particular, let n denote the number of parties. A non-reactive n-party functionality f is a (possibly randomized) mapping of n inputs to n outputs. A multiparty protocol with security parameter λ for computing a non-reactive functionality f is a protocol running in time $\mathrm{poly}(\lambda, n)$ and satisfying the following correctness requirement: if parties $P_1, \ldots, P_n$ with inputs $(x_1, \ldots, x_n)$ respectively, all run an honest execution of the protocol, then the joint distribution of the outputs $y_1, \ldots, y_n$ of the parties is statistically close to $f(x_1, \ldots, x_n)$. In this setting we assume that every party has access to a simultaneous broadcast channel where every party can simultaneously broadcast a message to all the other parties.

In the rest of this work, we denote an ℓ-round MPC protocol as $\pi = (\pi.\mathsf{Round}_1, \ldots, \pi.\mathsf{Round}_\ell, \pi.\mathsf{Out})$, where $\pi.\mathsf{Round}_j$, with $j \in [\ell]$ denotes the *next-message function* that takes as input all the messages generated by π in the rounds $1, \ldots, j-1$ (that we denote with τ_{j-1}) the state of the party P_i and outputs the message $\mathsf{msg}_{j,i}$. Additionally, we assume that all the parties run the same next message function algorithms (the only difference is the randomness and the input provided by each party). $\pi.\mathsf{Out}$ denotes the algorithm used to compute the final output of the protocol.

Defining Security. We assume that readers are familiar with standard simulation-based definitions of secure multi-party computation in the standalone setting. We provide a self-contained definition for completeness and refer to [Gol04] for a more complete description. The security of a protocol (with respect to a functionality f) is defined by comparing the real-world execution of the protocol with an ideal-world evaluation of f by a trusted party. More concretely, it is required that for every adversary $\mathcal{A}$, which attacks the real execution of the protocol, there exist an adversary $\mathcal{S}$, also referred to as a simulator, which can *achieve the same effect* in the ideal-world. Let us denote $\boldsymbol{x} = (x_1, \ldots, x_n)$.

The real execution In the real execution of the n-party protocol π for computing f is executed in the presence of an adversary $\mathcal{A}$. The honest parties follow the instructions of π. The adversary $\mathcal{A}$ takes as input the security parameter λ, the set $I \subset [n]$ of corrupted parties, the inputs of the corrupted parties, and an auxiliary input z. $\mathcal{A}$ sends all messages in place of corrupted parties and may follow an arbitrary polynomial-time strategy. The interaction of $\mathcal{A}$ with a protocol π defines a random variable $\mathrm{Real}_{\pi,\mathcal{A}(z),I}(k, \boldsymbol{x})$ whose value is determined by the coin tosses of the adversary and the honest players. This random variable contains the output of the adversary (which may be an arbitrary function of its view) as well as the outputs of the uncorrupted parties. We let $\mathrm{Real}_{\pi,\mathcal{A}(z),I}$ denote the distribution ensemble $\{\mathrm{Real}_{\pi,\mathcal{A}(z),I}(k, \boldsymbol{x})\}_{k\in\mathbb{N},\langle \boldsymbol{x},z\rangle\in\{0,1\}^*}$.

The ideal execution – security with abort. In this variant of the ideal model, fairness and output delivery are no longer guaranteed. This is the standard relaxation used when a strict majority of honest parties is not assumed. In this case, an ideal execution for a function f proceeds as follows:

- **Send inputs to the trusted party:** As before, the parties send their inputs to the trusted party, and we let x'_i denote the value sent by P_i.
- **Trusted party sends output to the adversary:** The trusted party computes $f(x'_1, \ldots, x'_n) := (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i\in I}$ to the adversary $\mathcal{S}$.
- **Adversary instructs trusted party to abort or continue:** This is formalized by having the adversary send either a continue or abort message to the trusted party. In the latter case, the trusted party sends to each uncorrupted party P_i its output value y_i. In the former case, the trusted party sends the special symbol $\perp$ to each uncorrupted party.
- **Outputs:** $\mathcal{S}$ outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

The interaction of $\mathcal{S}$ with the trusted party defines a random variable $\mathrm{Ideal}_{f,\mathcal{S}(z)}(k, \boldsymbol{x})$ as above. Having defined the real and the ideal world, we now proceed to define our notion of security.

Definition 20. *Let λ be the security parameter. Let f be an n-party randomized functionality, and π be an n-party protocol for $n \in \mathbb{N}$.*

We say that π securely realizes f in the presence of malicious adversaries if for every PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $I \subset [n]$ the following ensembles *are computational indistinguishable:*

$$\{\mathrm{Real}_{\pi,\mathcal{A}(z),I}(k,\boldsymbol{x})\}_{k\in\mathbb{N},\langle\boldsymbol{x},z\rangle\in\{0,1\}^*},\{\mathrm{Ideal}_{f,\mathcal{S}(z),I}(k,\boldsymbol{x})\}_{k\in\mathbb{N},\langle\boldsymbol{x},z\rangle\in\{0,1\}^*}.$$

3.6 Privacy with Knowledge of Outputs

We consider a relaxed notion of security known as *privacy with knowledge of outputs* [IKP10,PC12]. In this the input of the honest parties is protected in the standard simulation based sense, but the output of these parties might be incorrect. To formalize this notion we need to slightly modify the ideal execution as follows.

1. **Send inputs to the trusted party:** The parties send their inputs to the trusted party, and we let x'_i denote the value sent by P_i.
2. **Ideal functionality sends output to the adversary:** The ideal functionality computes $(y_1,\ldots,y_n) := f(x_1,\ldots,x_n)$ and sends $\{y_i\}_{i\in I}$ to the adversary $\mathcal{A}$.
3. **Output of the honest parties:** The adversary $\mathcal{S}$ sends either a continue or abort message or arbitrary values $\{y'_i\}_{i\in[n]\setminus I}$ to the ideal functionality. In the case of a continue message the ideal functionality sends y_i to the party P_i, in the case of an abort message every uncorrupted party receives $\perp$ and in the case that the ideal functionality receives arbitrary values $\{y'_i\}_{i\in[n]\setminus I}$ it forwards them to the honest parties.
4. **Outputs:** $\mathcal{S}$ outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

The interaction of $\mathcal{S}$ with the trusted party defines a random variable $\mathrm{Ideal}^{\mathsf{PKO}}_{f,\mathcal{S}(z)}(k,\boldsymbol{x})$ as above.

Having defined the real and the ideal world, we now proceed to define our notion of security.

Definition 21. *Let λ be the security parameter. Let f be an n-party randomized functionality, and π be an n-party protocol for $n\in\mathbb{N}$.*

We say that π securely realizes f with knowledge of outputs in the presence of malicious adversaries if for every PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $I\subset[n]$ the following ensembles are computational indistinguishable:

$$\{\mathrm{Real}_{\pi,\mathcal{A}(z),I}(k,\boldsymbol{x})\}_{k\in\mathbb{N},\langle\boldsymbol{x},z\rangle\in\{0,1\}^*},\{\mathrm{Ideal}^{\mathsf{PKO}}_{f,\mathcal{S}(z),I}(k,\boldsymbol{x})\}_{k\in\mathbb{N},\langle\boldsymbol{x},z\rangle\in\{0,1\}^*}.$$

In this work we also consider *single-output functionalities.* A single-output functionality f is such that $(y_1,\ldots,y_n) := f(x_1,\ldots,x_n)$ with $y_1 = y_2 = \cdots = y_n$ and $x_1,\ldots,x_n\in\{0,1\}^*$.

3.7 k-Signaling MPC

In this section we propose a new notion of MPC which we refer to as *k-Signaling MPC*. The definitions of secure MPC protocols that we have considered so far require the honest parties' inputs to be defined before the real (and ideal) world experiment. That is, in the *standard* definition of MPC, an honest party cannot start the protocol and decide its input at a later moment. We note that there are existing constructions of secure ℓ-round MPC protocols that do not require the input of the parties to compute the first $k-1$ rounds with $k\leq\ell-1$. However, the fact that a protocol does not need the input of the parties to compute the first $k-1$ rounds does not imply that the protocol retains its security in the case when the input of the honest parties is decided in the round k and might depend on the first $k-1$ rounds of the protocol (hence, it can be adversarially influenced). Clearly, if the inputs for the honest parties are decided by the adversary, we cannot hope to have any privacy on these inputs. Nonetheless, we could guarantee that these inputs are correctly used in the protocol (given that the honest parties follow the description of the protocol). In this paper we consider a definition of MPC where each party P_i has two inputs (x_i, w_i) and require that 1) x_i is protected in the standard simulation based sense but it has to be fixed before the real (ideal) world experiment starts (like in the standard definition of MPC) and 2) the input w_i can be decided by the adversary. For example, if the MPC protocol requires the input for the computation of the rounds $k, k+1, \dots, \ell$ then the adversary provides an input w_i to the party P_i at round $k-1$. For the value w_i our definition guarantees no privacy (indeed, it is decided by the adversary), but it guarantees that the input w_i, which is known by P_i, is properly used in the computation of the output of the functionality.

To see why this definition can be useful, consider the case where we use an ℓ-round MPC Π protocol as a building block for a more complicated protocol Π'. In Π' each party generates a message c_i using the randomness r_i (for example, c_i represents a commitment of a message m_i generated using the randomness r_i). Now, suppose that the parties want to compute a function f of their inputs $x_1, \dots, x_n$, the messages $c_1, \dots, c_n$ and the randomnesses $r_1, \dots, r_n$, but each party P_i wants to keep (x_i, r_i) private.

A simple solution to this problem would be to run the MPC protocol Π for the function f, where each party P_i runs the MPC protocol Π using the input $\big((x_i, r_i), (c_1, \dots, c_n)\big)$. The function f checks that all the parties have input the same list of messages $(c_1, \dots, c_n)$ and then performs a computation on the received inputs. We note that each party P_i has to input all the messages $(c_1, \dots, c_n)$ and not just c_i. Indeed, if this was the case then a corrupted party $P_j^\star$ could input any value $c_j^\star \neq c_j$ to get more information than it should on the private inputs of the honest parties. The problem with the above solution is that the protocol Π has to be run after the messages $(c_1, \dots, c_n)$ have been generated. This means that if the round complexity of Π is ℓ then the round complexity of Π' becomes $\ell' > \ell$ even if Π requires the inputs only for the computation of the rounds $\ell - 1$ and ℓ of Π.

In this scenario our definition becomes particularly useful. Indeed, we can parallelize the messages of Π with the other messages of Π' as long as each party P_i can specify the value (x_i, r_i) before the execution of the protocol starts, and the values $(c_1, \dots, c_n)$ at round $k-1$ (we recall that, at round k, Π requires all the inputs). We now provide a formal definition of our new notion and then show how to achieve it using a large class of MPC protocols in a round-preserving way.

The real execution. In the real execution the n-party protocol Π for computing f is executed in the presence of an adversary $\mathcal{A}$. The honest parties follow the instructions of Π. The adversary $\mathcal{A}$ takes as input the security parameter λ, the set $I \subset [n]$ of corrupted parties, the inputs of the corrupted parties, and an auxiliary input z. $\mathcal{A}$ sends all messages in place of corrupted parties and may follow an arbitrary polynomial-time strategy. At round $k-1$, $\mathcal{A}$ picks $w_i \in \{0,1\}^*$ and sends it to the honest party P_i for each $i \in [n] \setminus I$. Then each honest party P_i uses the input (x_i, w_i) to compute the rounds $k, k+1, \dots, \ell$ of Π. The adversary $\mathcal{A}$ continues its interaction with the honest parties following an arbitrary polynomial-time strategy.

The interaction of $\mathcal{A}$ with a protocol Π defines a random variable $\mathrm{Real}^{\mathsf{SigMPC}}_{\Pi,\mathcal{A}(z),I}(k, \boldsymbol{x})$ whose value is determined by the coin tosses of the adversary and the honest players. This random variable contains the output of the adversary (which may be an arbitrary function of its view) as well as the outputs of the uncorrupted parties. We let $\mathrm{Real}^{\mathsf{SigMPC}}_{\Pi,\mathcal{A}(z),I}$ denote the distribution ensemble $\{\mathrm{Real}^{\mathsf{SigMPC}}_{\Pi,\mathcal{A}(z),I}(k, \boldsymbol{x})\}_{k \in \mathbb{N}, \langle \boldsymbol{x}, z\rangle \in \{0,1\}^*}$.

The ideal execution

- **Send inputs to the trusted party:** Each honest party P_i sends $x_i = x_i'$ to the ideal functionality. The adversary sends $(x_j, w_j)_{j \in I}$ and $(w_i)_{i \in [n] \setminus I}$ to the ideal functionality.

Input: $(x_i^0, \mathsf{k}_i, x_i^1, \tau_i)_{i\in[n]}$.
If $\mathsf{Verify}(\mathsf{k}_i, x_i^1, \tau_i) = 0$ for any $i \in [n]$, then output $\perp$.
Compute $y_1, \dots, y_n := f'(x_1^0 || x_1^1, \dots, x_n^0 || x_n^1)$ and
set $y_i^0 := y_i^1 := y_i$ for all $i \in [n]$.
Output: (y_i^0, y_i^1) to the party P_i for each $i \in [n]$.

Fig. 11: Description of the function f.

- **Ideal functionality sends output to the adversary:** The ideal functionality computes $(y_1, \dots, y_n) := f(x_1\|w_1, \dots, x_n\|w_n)$ and sends $\{y_i\}_{i\in I}$ to the adversary $\mathcal{A}$ and w_i to P_i for each $i \in [n] \setminus I$.
- **Output of the honest parties:** The adversary $\mathcal{S}$ sends either a continue or abort message to the ideal functionality. In the case of a continue message the ideal functionality sends y_i to the party P_i, in the case of an abort message every uncorrupted party receives $\perp$.
- **Outputs:** $\mathcal{S}$ outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

The interaction of $\mathcal{S}$ with the trusted party defines a random variable $\mathrm{Ideal}^{\mathsf{SigMPC}}_{f,\mathcal{S}(z)}(k, \boldsymbol{x})$ as above. Having defined the real and the ideal world, we now proceed to define our notion of security.

Definition 22 (k-Signaling MPC). *Let λ be the security parameter. Let f be an n-party randomized functionality, and Π be an n-party ℓ-round protocol for $n, \ell \in \mathbb{N}$ where the input is required only to compute the rounds $k, \dots, \ell$ with $0 \leq k \leq \ell$.*

*We say that a protocol Π is k-*signaling *if it realizes f in the presence of malicious adversaries if for every PPT adversary with $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $I \subset [n]$ the following ensembles are computational indistinguishable:*

$$\{\mathrm{Real}^{\mathsf{SigMPC}}_{\pi,\mathcal{A}(z),I}(k, \boldsymbol{x})\}_{k\in\mathbb{N},\langle \boldsymbol{x},z\rangle\in\{0,1\}^*}, \{\mathrm{Ideal}^{\mathsf{SigMPC}}_{f,\mathcal{S}(z),I}(k, \boldsymbol{x})\}_{k\in\mathbb{N},\langle \boldsymbol{x},z\rangle\in\{0,1\}^*}.$$

From MPC protocols to k-Signaling MPC protocols. Our starting point is a $2n$-parties ℓ-round MPC protocol Π that does not require the input to compute the first $k-1$ rounds and a one-time MAC scheme $\mathsf{MAC} = (\mathsf{Setup}, \mathsf{Auth}, \mathsf{Verify})$. To construct our n-party k-Signaling MPC protocol we let each party to control two parties of Π, one that will be run on the private input and a MAC key (this information is known from the beginning of the protocol), and one that will be run on the signaled input (received in the end of round $k-1$ from the adversary) authenticated with the MAC key. The MPC protocol Π checks that the inputs are authenticated accordingly to the secret key, and if this check is successful then a function f over the secret and the signaled inputs of the parties is computed. The idea is that we can rely on the security of Π to claim that the private input is protected. To make sure that the correct signaled inputs are taken into account for the evaluation of the function, we rely on the fact that the signaled inputs are authenticated.

We describe the construction more formally and present a proof. Let f' be an n-input functionality and let Π be a $2n$-party MPC protocol that realizes the $2n$-input function f described in Fig. 11 with the property that it needs the input of the parties only to compute the rounds $k, k+1, \dots, \ell$ with $0 \leq k \leq \ell - 1$ where $\ell \in \mathbb{N}$ represents the round complexity of Π. Now, we construct a k-Signaling MPC protocol Π' that realizes f'. In our construction each party of Π' emulates two parties of Π. We denote with P_i^0 and P_i^1 the two parties (of the protocol Π) emulated by the party P_i (of the protocol Π'). Let x_i be the private input of P_i, then P_i performs the following steps.

1. Run the parties P_i^0 and P_i^1 until the round $k-1$ (recall that P_0^i and P_1^i do not need any input to compute the first $k-1$ rounds).
2. Upon receiving the signaled input w_i from the adversary, run Setup to sample a MAC key k_i and run P_i^0 using the input (x_i, k_i).
3. Compute $\tau_i \leftarrow \mathsf{Auth}(\mathsf{k}_i, w_i)$ and run P_i^1 using the input (w_i, τ_i).
4. When the protocol Π is over, P_i checks if the outputs of P_i^0 and P_i^1 correspond to the same value y_i. If that is the case then P_i outputs y_i, otherwise it aborts.

Theorem 1. *Let Π be an $2n$-party ℓ-round MPC protocol that securely realizes the function f of Fig. 11 and that requires the input only to compute the rounds $k, k+1, \dots, \ell$ with $0 \leq k \leq \ell - 1$ and let $\mathsf{MAC} = (\mathsf{Setup}, \mathsf{Auth}, \mathsf{Verify})$ be a one-time MAC secure scheme, then Π is an n-party ℓ-round k-Signaling MPC protocol that securely realizes the function f'.*

Proof. Let $\mathcal{A}'$ be the adversary attacking our protocol Π'. To simplify the description of the proof we assume that only one party P_i is honest. The proof can be easily extended to the case where more than one party is honest. To prove the security of our protocol we need to describe a simulator $\mathcal{S}'$. Before doing that, we define an augmented machine $\mathcal{M}$ that works as described in Fig. 12. $\mathcal{M}$ internally runs the adversary $\mathcal{A}'$ and computes the messages for the party P_i^1. In addition, all the messages received by the adversary $\mathcal{A}'$ and the messages computed by P_i^1 are forwarded to the external interface of $\mathcal{M}$ which we refer as the *left-session*, and all the messages that come from the left session are forwarded to the adversary $\mathcal{A}'$. In a nutshell, $\mathcal{M}$ acts as an adversary for the protocol Π where only the party P_i^0 is honest.

```
$\underline{\mathcal{M}(\mathcal{A}', r'_{\mathcal{A}}, \mathsf{k}_i):}$
Run $\mathcal{A}'$ using the randomness $r'_{\mathcal{A}}$
For each Round $r = 1, \dots, k-1$
    1. Upon receiving the message $\mathsf{msg}_{r,i,0}$ from the left session, run $P_i^1$ thus obtaining the message
       $\mathsf{msg}_{r,i,1}$ and send $(\mathsf{msg}_{r,i,0}, \mathsf{msg}_{r,i,1})$ to $\mathcal{A}'$.
    2. Upon receiving the messages $\{\mathsf{msg}_{r,j,b}\}_{j \in I, b \in \{0,1\}}$ from $\mathcal{A}'$ send $\{\mathsf{msg}_{r,j,b}\}_{j \in I, b \in \{0,1\}} \cup \{\mathsf{msg}_{r,i,1}\}$
       in the left session.
Round $k$
    1. Upon receiving the message $\mathsf{msg}_{k,i,0}$ from the left session and upon receiving $w_i$ from $\mathcal{A}'$, compute
       $\tau_i \leftarrow \mathsf{Auth}(\mathsf{k}_i, w_i)$ and run $P_i^1$ on input $(w_i, \tau_i)$ thus obtaining the message $\mathsf{msg}_{k,i,1}$ and send
       $(\mathsf{msg}_{k,i,0}, \mathsf{msg}_{k,i,1})$ to $\mathcal{A}'$.
    2. Upon receiving the messages $\{\mathsf{msg}_{k,j,b}\}_{j \in I, b \in \{0,1\}}$ from $\mathcal{A}'$ send $\{\mathsf{msg}_{k,j,b}\}_{j \in I, b \in \{0,1\}} \cup \{\mathsf{msg}_{k,i,1}\}$
       in the left session.
For each Round $r = k+1, \dots, \ell$
    1. Upon receiving the message $\mathsf{msg}_{r,i,0}$ from the left session, run $P_i^1$ thus obtaining the message
       $\mathsf{msg}_{r,i,1}$ and send $(\mathsf{msg}_{r,i,0}, \mathsf{msg}_{r,i,1})$ to $\mathcal{A}'$.
    2. Upon receiving the messages$\{\mathsf{msg}_{r,j,b}\}_{j \in I, b \in \{0,1\}}$ from $\mathcal{A}'$ send $\{\mathsf{msg}_{r,j,b}\}_{j \in I, b \in \{0,1\}} \cup \{msg_{r,i,1}\}$
       in the left session.
```

Fig. 12: The augmented machine $\mathcal{M}$ which emulates the adversary for Π.

By assumption, we know that for every PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any $I \subset [2n]$ the following ensembles are computational indistinguishable:

$$\{\mathrm{Real}_{\Pi,\mathcal{A}(z),I}(k, \boldsymbol{x})\}_{k \in \mathbb{N}, \langle \boldsymbol{x}, z \rangle \in \{0,1\}^*}, \{\mathrm{Ideal}_{f,\mathcal{S}(z),I}(k, \boldsymbol{x})\}_{k \in \mathbb{N}, \langle \boldsymbol{x}, z \rangle \in \{0,1\}^*} \ .$$

Hence, there exists a simulator $\mathcal{S}$ for the adversary $\mathcal{M}(\mathcal{A}', \cdot, \cdot)$. At a high level, our simulator $\mathcal{S}'$ internally runs $\mathcal{S}$, and its formal description is provided in Fig. 13.

$\underline{\mathcal{S}'}$

- Sample a MAC key k_i.
- Sample a sufficiently long random R and run $\mathcal{S}$ for the adversary $\mathcal{M}(\mathcal{A}, R, \mathsf{k}_i)$.
- For every query $\big((x_1^0, \mathsf{k}_1), (x_1^1, \tau_1), \ldots, (x_i^1, \tau_i), \ldots, (x_n^0, \mathsf{k}_n), (x_n^1, \tau_n)\big)$ issued by $\mathcal{S}$ do the following:
  1. For each $j \in [n]$ check if $\mathsf{Verify}(\mathsf{k}_j, x_j^1, \tau_j) = 0$ then return $\perp$ to $\mathcal{S}$, else continue as follows.
  2. Query the ideal functionality f using $\big((x_1^0, x_1^1), \ldots, x_i^1, \ldots, (x_n^0, x_n^1)\big)$, and receive $\big((y_1^0, y_1^1), \ldots, (y_n^0, y_n^1)\big) := f(x_1^0, x_1^1, \ldots, x_n^0, x_n^1)$ as an output.
  3. For each i set $y_i := y_i^0 := y_i^1$.
  4. Send $(y_1, \ldots, y_n)$ to $\mathcal{S}$.
- When $\mathcal{S}$ stops, output what $\mathcal{S}$ outputs.

Fig. 13: The simulator $\mathcal{S}'$ for our k-Signaling MPC protocol Π'.

Suppose by contradiction that our theorem does not hold. That is, suppose that there exists an adversary $\mathcal{A}'$ that breaks the security of Π', then we can construct an adversary $\mathcal{M}(\mathcal{A}', \cdot, \cdot)$ that breaks the security of Π. We note that the only other reason why the simulator $\mathcal{S}'$ could fail is because $\mathcal{S}$ extracts a couple of value (x_i^1, τ_i) such that $\mathsf{Verify}(\mathsf{k}_i, x_i^1, \tau_i) = 1$ and $x_i^1 \neq w_i$ (where w_i is the values sent by $\mathcal{A}'$). If this is the case, then we can construct a reduction to break the security of the one-time MAC.

*N*ote: the use of MAC seems redundant as it might be possible to argue that the simulator $\mathcal{S}$ always extracts the signaled input, indeed we believe that the proof would work without relying on a one-time MAC. However, given that the use of one-time MAC makes the proof more intuitive and that one-time MACs can be instantiated information theoretically we have chosen to provide the above proof.

□

4 Our Compiler: Circuit-Scalable MPC

In this section we prove our main theorems on how to construct a circuit-scalable MPC protocol that realizes any functionality f with privacy with knowledge of outputs. We refer to Section 2 for a simplified description of the protocol for the two-party case and to Fig. 14 for the formal description of our compiler.

Our Construction makes use of the following cryptographic tools:

- An ℓ-round k-signaling MPC protocol $\Pi^{\mathsf{M}} = (\Pi^{\mathsf{M}}.\mathsf{Round}_1, \ldots, \Pi^{\mathsf{M}}.\mathsf{Round}_\ell, \Pi^{\mathsf{M}}.\mathsf{Out})$ (not necessarily communication efficient) with $k \geq 3$ that securely evaluates the function C_{ct} (described in Fig. 15), where $\Pi^{\mathsf{M}}.\mathsf{Round}_k$ takes the input of the party P_i that we denote with y_i.[14] In the description of our compiler we assume, without loss of generality, that Π^{M} is 3-signaling.[15]
- A strong succinct single-key simulation secure decomposable FE combiner $\mathsf{DFEC} = (\mathsf{DFEC.Setup}, \mathsf{DFEC.Enc}, \mathsf{DFEC.KeyGen}, \mathsf{DFEC.Dec}, \mathsf{DFEC.Partition})$ for n FE candidates.
- A non-interactive computationally hiding commitment scheme Com.

[14] To simplify the description of the protocol we assume that the entire input (the signaled part and the private part) is provided in round k.

[15] It is easy to see that any k'-signaling MPC with $k' > 3$ can be turned into a 3-signaling MPC protocol since that the signaled input received in round 2 can be ignore up to round $k' - 1$.

$\underline{\Pi^{\mathsf{FE}}}$

Initialization: Each $i \in [n]$ party P_i has input $x_i \in \{0,1\}^*$ as its secret input. We initialize τ_0 to 1^λ.

Round 1.

1. Compute $\mathsf{msg}_{1,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Round}_1(1^\lambda)$.
2. Sample $r_{\mathsf{Setup}}^{i\to k}, r_{\mathsf{com}}^{i\to k} \leftarrow \{0,1\}^\lambda$ for all $k \in [n]$, compute $\mathsf{com}_{\mathsf{Setup}}^{i\to k} := \mathsf{Com}(r_{\mathsf{Setup}}^{i\to k}; r_{\mathsf{com}}^{i\to k})$ and set $\mathsf{com}_{\mathsf{Setup}}^{i} := \{\mathsf{com}_{\mathsf{Setup}}^{i\to k}\}_{k\in[n]}, \mathsf{open}_{\mathsf{Setup}}^{i} := (r_{\mathsf{Setup}}^{i\to k}, r_{\mathsf{com}}^{i\to k})_{k\in[n]}$.
3. Send $(\mathsf{msg}_{1,i}, \mathsf{com}_{\mathsf{Setup}}^{i})$.

Round 2.

1. Let τ_1 denote the transcript of the protocol Π^{M} up to round 1.
2. Compute $\mathsf{msg}_{2,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Round}_2(\tau_1)$.
3. Send $(\mathsf{msg}_{2,i}, (r_{\mathsf{Setup}}^{i\to j})_{j\in[n]\setminus\{i\}})$.

Round 3.

1. Let τ_2 denote the transcript of the protocol Π^{M} up to round 2.
2. Compute $\mathsf{msg}_{3,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Round}_3(y_i, \tau_2)$, where $y_i := (x_i, \mathsf{com}_{\mathsf{Setup},i}, \mathsf{open}_{\mathsf{Setup}}^{i}, R_{\mathsf{Setup}}^{i}, r_i^{\mathsf{Enc}})$, $\mathsf{com}_{\mathsf{Setup},i} := \{\mathsf{com}_{\mathsf{Setup}}^{k}\}_{k\in[n]}$, $R_{\mathsf{Setup}}^{i} := (r_{\mathsf{Setup}}^{j\to k})_{j\in[n],k\in[n]\setminus\{j\}}$ and $r_i^{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$.
3. Send $\mathsf{msg}_{3,i}$.

For each round $k \in \{4,\ldots,\ell-1\}$.

1. Let τ_{k-1} denote the transcript of the protocol Π^{M} up to round $k-1$.
2. Compute the second round message $\mathsf{msg}_{k,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Round}_k(\tau_{k-1})$.
3. Send $\mathsf{msg}_{k,i}$.

Round ℓ.

1. Let $\tau_{\ell-1}$ denote the transcript of the protocol Π^{M} up to round $\ell-1$.
2. Compute $r_i^{\mathsf{Setup}} = \bigoplus_{k\in[n]} r_{\mathsf{Setup}}^{k\to i}$.
3. Generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$, compute the partition of C, i.e. $(C_1 \ldots, C_n) \leftarrow \mathsf{DFEC.Partition}(1^\lambda, C)$ and generate $\mathsf{sk}_i \leftarrow \mathsf{FE}_i.\mathsf{KeyGen}(\mathsf{msk}_i, C_i; r_i^{\mathsf{KeyGen}})$ with $r_i^{\mathsf{KeyGen}} \leftarrow \{0,1\}^\lambda$.
4. Compute the fourth round message $\mathsf{msg}_{\ell,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Round}_\ell(\tau_{\ell-1})$.
5. Send $(\mathsf{msg}_{\ell,i}, \mathsf{sk}_i)$.

Output Computation.

1. Let τ_ℓ denote the transcript of the protocol Π^{M} up to round ℓ.
2. Compute the output of Π^{M} as $(\mathsf{ct}, (\tilde{r}_{\mathsf{Setup}}^{k\to i})_{i\in[n],k\in[n]\setminus\{i\}}) \leftarrow \Pi^{\mathsf{M}}.\mathsf{Out}(\tau_\ell)$.
3. Check that $(\tilde{r}_{\mathsf{Setup}}^{k\to i})_{i\in[n],k\in[n]\setminus\{i\}}$ is equal to $(r_{\mathsf{Setup}}^{k\to i})_{i\in[n],k\in[n]\setminus\{i\}}$, if not then **Abort**.
4. Output $\mathsf{DFEC.Dec}(\mathsf{sk}_C, \mathsf{ct})$ with $\mathsf{sk}_C = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.

Fig. 14: Description of the protocol Π^{FE} that securely realizes any functionality with knowledge of outputs.

Input: $(x_i, \mathsf{com}_{\mathsf{Setup},i}, \mathsf{open}_{\mathsf{Setup}}^{i}, R_{\mathsf{Setup}}^{i}, r_i^{\mathsf{Enc}})_{i\in[n]}$

- Check that $\mathsf{com}_{\mathsf{Setup},i} = \mathsf{com}_{\mathsf{Setup},j}$ for all $i,j \in [n]$.
- Check that $R_{\mathsf{Setup}}^{i} = R_{\mathsf{Setup}}^{j}$ for all $i,j \in [n]$.
- Parse $\mathsf{com}_{\mathsf{Setup},i}$ as $\{\mathsf{com}_{\mathsf{Setup}}^{k}\}_{k\in[n]}$ and $\mathsf{com}_{\mathsf{Setup}}^{i}$ as $\{\mathsf{com}_{\mathsf{Setup}}^{i\to k}\}_{k\in[n]}$ for each $i \in [n]$.
- Parse R_{Setup}^{i} as $(\tilde{r}_{\mathsf{Setup}}^{j\to k})_{j\in[n],k\in[n]\setminus\{j\}}$.
- For all $i \in [n]$, parse $\mathsf{open}_{\mathsf{Setup}}^{i}$ as $(r_{\mathsf{Setup}}^{i\to j}, r_{\mathsf{com}}^{i\to j})_{j\in[n]}$
- For all $i,j \in [n]$ check that $\mathsf{com}_{\mathsf{Setup}}^{i\to j} = \mathsf{Com}(r_{\mathsf{Setup}}^{i\to j}; r_{\mathsf{com}}^{i\to j})$
- For all $i \in [n], j \in [n] \setminus \{i\}$ check that $\tilde{r}_{\mathsf{Setup}}^{i\to j} = r_{\mathsf{Setup}}^{i\to j}$

If one of the above checks fails then output $\perp$, continue as follows otherwise.

For each $i \in [n]$, compute $r_i^{\mathsf{Setup}} = \bigoplus_{k\in[n]} r_{\mathsf{Setup}}^{k\to i}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$

Let $\mathsf{msk} := (\mathsf{msk}_1, \ldots, \mathsf{msk}_n), x = (x_1, \ldots, x_n), r^{\mathsf{Enc}} := \bigoplus_{i\in[n]} r_i^{\mathsf{Enc}}$.

Output: $\mathsf{ct} := \mathsf{DFEC.Enc}(\mathsf{msk}, x; r^{\mathsf{Enc}})$ and $\{r_{\mathsf{Setup}}^{k\to j}\}_{j\in[n],k\in[n]\setminus\{j\}}$ to P_i.

Fig. 15: Circuit C_{ct}

Theorem 2. *Let* DFEC *be a single-key simulation secure decomposable FE combiner with circuit size* cs_{Setup} *the setup algorithm* $\mathsf{DFEC.Setup}$*, circuit size* cs_{ct} *for the encryption algorithm* $\mathsf{DFEC.Enc}$ *and functional key size* s_{sk}*, let* Com *be a commitment scheme and let* Π^{M} *be an* ℓ*-round MPC protocol that securely realizes* C_{ct}*, then* Π^{FE} *is an* ℓ*-round MPC protocol that realizes the single-output functionality* C *with knowledge of outputs which has communication complexity* $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, s_{\mathsf{sk}})$*.*

We split this theorem into two Lemmas and prove them separately:

Lemma 1. *Let* DFEC *be a single-key simulation secure decomposable FE combiner with circuit size* cs_{Setup} *for the setup algorithm* $\mathsf{DFEC.Setup}$*, circuit size* cs_{Enc} *for the encryption algorithm* $\mathsf{DFEC.Enc}$ *and functional key size* s_{sk}*, then* Π^{FE} *has communication complexity* $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, s_{\mathsf{sk}})$*.*

Proof. We divide the analysis of the communication complexity into two steps. In the first step, we analyze the communication complexity of the inner MPC protocol Π^{M} and in the second step the communication complexity of the additional values sent in the outer MPC protocol.

We remark that the operations executed inside the MPC protocol Π^{M}, besides the generation of the master secret keys and the encryption, have communication complexity $\mathrm{poly}(\lambda, n)$. Since the circuit size of the setup algorithm is cs_{Setup} and the circuit size of the encryption algorithm is cs_{Enc} the resulting message length is $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}})$, i.e. $|\mathsf{msg}_{k,i}| = \mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}})$ for all $k \in \{1, \ldots, \ell\}$ and all $i \in [n]$.

After analyzing the communication complexity of the inner MPC protocl Π^{M}, we continue with the analysis of the messages that are sent in addition to the messages of Π^{M}.

The additional messages, strings of length λ and commitments, that are output by every party P_i for all $i \in [n]$ in round k, with $k \in \{1, \ldots, \ell - 1\}$ are of length $\mathrm{poly}(\lambda, n)$. Since the additional output in round ℓ contains of the secret key sk_i for all $i \in [n]$ and therefore increases in size s_{sk}. This results in a communication complexity of $\mathrm{poly}(\lambda, n, s_{\mathsf{sk}})$ for the additional messages.

Combining the two analysis yields an overall communication complexity of $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, s_{\mathsf{sk}})$. □

To specify the values $cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}$ and s_{sk} we apply the definition of combiner succinctness (Definition 7), which results in the fact that $cs_{\mathsf{Enc}} = \mathrm{poly}(\lambda, n, d, L_{\mathsf{out}})$ and $s_{\mathsf{sk}} = \mathrm{poly}(\lambda, n, d, L_{\mathsf{in}})$. For the circuit size cs_{Setup} of the setup algorithm Setup it holds that $cs_{\mathsf{Setup}} = \mathrm{poly}(\lambda, n)$. This results in an overall communication complexity of $\mathrm{poly}(\lambda, n, d, L_{\mathsf{in}}, L_{\mathsf{out}})$.

Lemma 2. *Let* DFEC *be a single-key simulation secure decomposable FE combiner, and let* Π^{M} *be a* k*-signaling* ℓ*-round MPC protocol (with* $k \geq 3$*) that securely realizes* C_{ct}*, then* Π^{FE} *is an* ℓ*-round MPC protocol that realizes the single-output functionality* C *with privacy with knowledge of outputs.*

Proof. To prove our lemma we need to show that for every PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $I \subset [n]$ the following quantity is negligible:

$$|\Pr[\mathrm{Real}_{\Pi^{\mathsf{FE}}, \mathcal{A}(z), I}(\lambda, \boldsymbol{x}) = 1] - \Pr[\mathrm{Ideal}^{\mathsf{PKO}}_{C, \mathcal{S}(z), I}(\lambda, \boldsymbol{x}) = 1]| \ ,$$

where $\boldsymbol{x} = \{x_i\}_{i \in [n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$.

Also in this case, for simplicity, we assume that all but one of the parties are corrupted, where we denote the set that contains the indices of all the corrupted parties as I, i.e. $|I| = n - 1$. Before describing how $\mathcal{S}$ works, we define an algorithm $\mathcal{M}$ that we refer to as the *augmented machine*. The augmented machine internally runs the adversary $\mathcal{A}$ (we refer to this as the right session), and acts as a proxy between $\mathcal{A}$ and its external interface with respect to the messages of $\mathit{\Pi}^{\mathsf{M}}$ (we refer to this as the left session). To describe our simulator we then need to described the augmented machine $\mathcal{M}$ and how $\mathcal{S}$ interacts with it (i.e., how the messages of $\mathit{\Pi}^{\mathsf{M}}$ are computed).

The reason why we describe our simulator via the augmented machine $\mathcal{M}$ is to deal with the rewinds that the simulator of $\mathit{\Pi}^{\mathsf{M}}$ might do. We note that $\mathcal{M}$ acts as an adversary for the protocol $\mathit{\Pi}^{\mathsf{M}}$. Hence, we can consider the simulator $\mathit{\Pi}^{\mathsf{M}}.\mathcal{S}$ for $\mathit{\Pi}^{\mathsf{M}}$ (which exists by assumption) for the adversary $\mathcal{M}$. Our simulator $\mathcal{S}$ will then simply run $\mathit{\Pi}^{\mathsf{M}}.\mathcal{S}$. For the formal description of $\mathcal{M}$ we refer to Fig. 16 and for the formal description of $\mathcal{S}$ we refer to Fig. 17.

$\underline{\mathcal{M}(r_{\mathcal{A}}):}$

Round 1.

1. Receive the message $\mathsf{msg}_{1,i}$ in the left session.
2. Sample $r^{i\to k}_{\mathsf{Setup}}, r^{i\to k}_{\mathsf{com}}$ for all $k \in [n]$, compute $\mathsf{com}^{i\to k}_{\mathsf{Setup}} = \mathsf{Com}(r^{i\to k}_{\mathsf{Setup}}; r^{i\to k}_{\mathsf{com}})$, set $\mathsf{com}^{i}_{\mathsf{Setup}} := \{\mathsf{com}^{i\to k}_{\mathsf{Setup}}\}_{k\in[n]}$ and $\mathsf{open}^{i}_{\mathsf{Setup}} := (r^{i\to k}_{\mathsf{Setup}}, r^{i\to k}_{\mathsf{com}})_{k\in[n]}$.
3. Send $(\mathsf{msg}_{1,i}, \mathsf{com}^{i}_{\mathsf{Setup}})$ in the right session.
4. Receive $(\mathsf{msg}_{1,j}, \mathsf{com}^{j}_{\mathsf{Setup}})_{j\in I}$ as a reply in the right session and output $\{\mathsf{msg}_{1,j}\}_{j\in I}$ in the left session.

Round 2.

1. Receive the message $\mathsf{msg}_{2,i}$ in the left session.
2. Sample random values $r^{i\to j}_{\mathsf{Setup}'}$.
3. Output $\mathsf{msg}_{2,i}, \{r^{i\to j}_{\mathsf{Setup}'}\}_{j\in I}$ in the right session.
4. Receive $(\mathsf{msg}_{2,j}, r^{j\to i}_{\mathsf{Setup}})_{j\in I}$ as a reply in the right session and output $\{\mathsf{msg}_{2,j}\}_{j\in I}$ in the left session.

For each round $k \in \{3, \ldots, \ell - 1\}$.

1. Upon receiving the message $\mathsf{msg}_{k,i}$ from the left session forward it to $\mathcal{A}$.
2. Receive the messages $\{\mathsf{msg}_{k,j}\}_{j\in I}$ and forward them in the left session.

Round ℓ.

1. Receive the message $\mathsf{msg}_{\ell,i}$ in the left session.
2. Compute $r^{\mathsf{Setup}}_{i} = \bigoplus_{k\in[n]\setminus\{i\}} r^{k\to i}_{\mathsf{Setup}} \oplus r^{i\to i}_{\mathsf{Setup}'}$ with a random value $r^{i\to i}_{\mathsf{Setup}'}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^{\lambda}; r^{\mathsf{Setup}}_{i})$, compute the partition of C, i.e. $(C_1 \ldots, C_n) \leftarrow \mathsf{DFEC.Partition}(1^{\lambda}, C)$ and generate $\mathsf{sk}_i \leftarrow \mathsf{FE}_i.\mathsf{KeyGen}(\mathsf{msk}_i, C_i; r^{\mathsf{KeyGen}}_{i})$ for a random r^{KeyGen}_{i}.
3. Send $(\mathsf{msg}_{\ell,i}, \mathsf{sk}_i)$ in the right session.
4. Receive $(\mathsf{msg}_{\ell,j})_{j\in I}$ as a reply in the right session and output $\{\mathsf{msg}_{\ell,j}\}_{j\in I}$ in the left session.

Fig. 16: The augmented machine $\mathcal{M}$ which emulates the adversary for $\mathit{\Pi}^{\mathsf{M}}$.

$\underline{\mathcal{S}}$

- Sample a sufficiently long random R and run the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ for the adversary $\mathcal{M}$.
- For every query $(x_j, \mathsf{com}_{\mathsf{Setup},j}, \mathsf{open}^j_{\mathsf{Setup}}, R^j_{\mathsf{Setup}}, r^{\mathsf{Enc}}_j)_{j \in I}$ and $\mathsf{com}_{\mathsf{Setup},i}, R^i_{\mathsf{Setup}}$ issued by $\Pi^{\mathsf{M}}.\mathcal{S}$ do the following:
  1. Check that $\mathsf{com}_{\mathsf{Setup},i} = \mathsf{com}_{\mathsf{Setup},j}, R^i_{\mathsf{Setup}} = R^j_{\mathsf{Setup}}$ for all $j \in I$, $\mathsf{com}^{j \to k}_{\mathsf{Setup}} = \mathsf{Com}(r^{j \to k}_{\mathsf{Setup}}; r^{j \to k}_{\mathsf{com}})$ with $\mathsf{open}^j_{\mathsf{Setup}} := (r^{j \to k}_{\mathsf{Setup}}, r^{j \to k}_{\mathsf{com}})_{k \in [n]}$ for all $j \in I, k \in [n]$ and $\tilde{r}^{j \to k}_{\mathsf{Setup}} = r^{j \to k}_{\mathsf{Setup}}$ with $R^i_{\mathsf{Setup}} := (\tilde{r}^{j \to k}_{\mathsf{Setup}})_{j \in [n], k \in [n] \setminus \{j\}}$ for all $j \in I, k \in [n] \setminus \{j\}$. If one of these checks fails, **Abort**.
  2. Compute $r^{\mathsf{Setup}}_i = \bigoplus_{k \in [n] \setminus \{i\}} r^{k \to i}_{\mathsf{Setup}} \oplus r^{i \to i}_{\mathsf{Setup}'}$ with a random value $r^{i \to i}_{\mathsf{Setup}'}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_i)$.
  3. Query the ideal functionality C using $\{x_j\}_{j \in I}$ and receive $C(x_1, \ldots, x_n)$ as an output.
  4. Compute $r^{\mathsf{Setup}}_j = \bigoplus_{k \in [n] \setminus \{i\}} r^{k \to j}_{\mathsf{Setup}} \oplus r^{i \to j}_{\mathsf{Setup}'}$ for all $j \in I$ and generate $\mathsf{msk}_j \leftarrow \mathsf{FE}_j.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_j)$. Simulate a ciphertext $\mathsf{ct} \leftarrow \mathsf{DFEC}.\mathcal{S}(\{\mathsf{msk}_i\}_{i \in [n]}, C, C(x_1, \ldots, x_n), I)$ and send $\mathsf{ct}^* := (\mathsf{ct}, \{r^{k \to l}_{\mathsf{Setup}}\}_{k \in [n] \setminus \{i\}, l \in [n] \setminus \{k\}}, \{r^{i \to l}_{\mathsf{Setup}'}\}_{l \in [n] \setminus \{i\}})$ to $\Pi^{\mathsf{M}}.\mathcal{S}$.
- When $\Pi^{\mathsf{M}}.\mathcal{S}$ stops, output what $\Pi^{\mathsf{M}}.\mathcal{S}$ outputs.

Fig. 17: The simulator $\mathcal{S}$ for our protocol Π.

We describe the hybrid experiments that we use to prove the lemma. We first give an informal description:

Hybrid H_0: Hybrid H_0 is identical to the real world.

Hybrid H_1: In hybrid H_1, the messages of the inner MPC protocol Π^{M} are is simulated using the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ (which exists by assumption). The transition between hybrid H_0 and H_1 is justified by the malicious security of the MPC protocol Π^{M} and formally proven in Claim 1.

Hybrid H_2: In hybrid H_2, commitments $(\mathsf{com}^{i \to j}_{\mathsf{Setup}})_{\mathsf{Setup}}\}_{j \in I}$ commit to the values $(r^{i \to j}_{\mathsf{Setup}})_{j \in I}$, but the values output in the second round and used to complete the remaining rounds are freshly generated random values $\{r^{i \to j}_{\mathsf{Setup}'}\}_{j \in I}$. The transition between hybrid H_1 and H_2 is justified by the hiding property of the commitment Com and formally proven in Claim 2.[16]

Hybrid H_3: In hybrid H_3, the randomness used to generate the master secret key msk_i does is computed using the randomness $r^{\mathsf{Setup}}_i = \bigoplus_{k \in [n] \setminus \{i\}} r^{k \to i}_{\mathsf{Setup}} \oplus r^{i \to i}_{\mathsf{Setup}'}$ where $r^{i \to i}_{\mathsf{Setup}'}$ is a randomly sample value which is different from the $r^{i \to i}_{\mathsf{Setup}}$ committed to in the first round. The transition between hybrid H_2 and H_3 is justified by the hiding property of the commitment Com and formally proven in Claim 3.

Hybrid H_4: Hybrid H_4 is identical to the ideal world. In this hybrid, the honestly generated ciphertext ct is replaced by a simulated ciphertext that is generated using the simulator $\mathsf{DFEC}.\mathcal{S}$ of the functional encryption combiner DFEC. The transition between H_3 and H_4 is justified by the succinct single-key simulation security of the functional encryption combiner DFEC and requires the introduction of two intermediate hybrids $\mathsf{H}^\star_3$ and $\mathsf{H}^\star_4$, which are described below. This hybrid is described in more detail on Page 33.

We also need to introduce the intermediate hybrids $\mathsf{H}^\star_3$ and $\mathsf{H}^\star_4$:

Hybrid $\mathsf{H}^\star_3$: The hybrid $\mathsf{H}^\star_3$, is an intermediate hybrid that works exactly as hybrid H_3 with the only difference that look ahead threads for the second and third round are created and freshly sampled random values $\{r^{i \to j}_{\mathsf{Setup}''}\}_{j \in I}$ instead of $\{r^{i \to j}_{\mathsf{Setup}'}\}_{j \in I}$ are output in the second round of the main thread. Since the values $\{r^{i \to j}_{\mathsf{Setup}'}\}_{j \in I}$ and $\{r^{i \to j}_{\mathsf{Setup}''}\}_{j \in I}$ are randomly sampled, the output distribution of the messages in the second round is the same and therefore the hybrids H_3 and $\mathsf{H}^\star_3$ are perfectly indistinguishable. This hybrid is described in more detail on Page 32.

[16] We make use of the fact here that commitments are secure under parallel composition, as mentioned after Definition 17, and output a new random value to all the parties P_j with $j \in I$.

Hybrid $\mathsf{H}_4^\star$: In this hybrid, the same look ahead threads as in $\mathsf{H}_3^\star$ are created, but the honestly generated ciphertext ct is replaced by a simulated ciphertext that is generated using the simulator of the functional encryption combiner $\mathsf{DFEC}.\mathcal{S}$. The transition between the hybrids is proven by relying on the succinct single-key simulation security of the functional encryption combiner DFEC. To enforce that the master secret keys the corrupted parties generate match the master secret keys that are generated by the challenger, we need to sample the values $\{r^{i \to j}_{\mathsf{Setup}''}\}_{j \in I}$ such that their XOR with the values output by the other parties in the second round $\{r^{j \to k}_{\mathsf{Setup}}\}_{j \in I, k \in [n] \setminus \{i\}}$ match the randomness used by the challenger for the master secret key generation. This transition is formally proven in Claim 4.

For the formal description of the hybrids, we also use an augmented machine. More precisely, we define a different augmented machine for each hybrid experiment. In addition, for each hybrid experiment H_k with $k \in \{0, \ldots, 4\}$ we need to specify how to construct the answers to the query made by the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$. For the formal description of the augmented machines we refer to Fig. 19, whereas the formal description of the hybrid experiments is provided in Fig. 18.

□

Claim 1 (Transition from H_0 to H_1) *Let Π^{M} be a maliciously secure MPC protocol, then the output distributions of the hybrid experiments H_0 and H_1 are computationally indistinguishable.*

Proof. By assumption we know that for every PPT adversary $\mathcal{A}'$ there exists a PPT adversary $\mathcal{S}'$ such that for any $I \subset [n]$ the following quantity is negligible:

$$|\Pr[\mathrm{Real}_{\Pi^{\mathsf{M}},\mathcal{A}(z),I}(k, \boldsymbol{x}) = 1] - \Pr[\mathrm{Ideal}_{C_{\mathsf{ct}},\mathcal{S}'(z),I}(k, \boldsymbol{x}) = 1]|$$

where $\boldsymbol{x} = \{x_i\}_{i \in [n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$.

Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids with non-negligible probability then we can use the adversary $\mathcal{A}' := \mathcal{M}_0(\mathcal{A}, \cdot)$ to break the security of Π^{M}. The description of the augmented machine $\mathcal{M}_0$ can be found in Fig. 19.

Note that $\mathcal{M}_0(\mathcal{A}, \cdot)$ is a valid adversary for Π^{M} as, in each round $k \in [\ell]$ it waits to receive the messages of Π^{M} generated bye the honest party P_i and replies with the messages computed by the malicious parties indexed by $[n] \setminus \{i\}$. In the reduction we have a challenger that, having black box access to $\mathcal{M}_0(\mathcal{A}, \cdot)$, either interacts with it using the messages of Π^{M} generated accordingly to the honest procedure or using the simulator $\mathcal{S}'$, which exists by the security definition. We note that in the case where the messages are generated accordingly to the honest procedure that the output of $\mathcal{M}_0(\mathcal{A}, \cdot)$ corresponds to the output of H_0, otherwise it corresponds to the output of H_0.

□

Before we continue with the description of the transition between the remaining hybrids, we need to distinguish between two different cases. The case where the adversary aborts in round three with probability $1 - \mathrm{negl}(\lambda)$, and the case where the adversary completes the third round with some non-negligible probability. In the case that the adversary aborts in the third round, the following hybrids are not necessary, since the security in this case can be directly reduced to the security of the inner MPC protocol Π^{M}. Indeed, note that the input of the honest parties appears only in the messages of Π^{M} and nowhere else.

In the case that the adversary completes the third round with some non-negligible probability, the proof continues as follows.

Claim 2 (Transition from H_1 to H_2) *If Com is a computationally hiding commitment scheme, then the output distribution of the hybrids H_1 and H_2 are computationally indistinguishable.*

$\mathsf{H}_0, \boxed{\mathsf{H}_1, \boxed{\mathsf{H}_2}, \mathsf{H}_3, \mathsf{H}_4}$

1. Run Π^{M} against the adversary $\mathcal{M}_0(\mathcal{A})$. Upon receiving $\mathsf{com}_{\mathsf{Setup},i}$ in round 1 and R^i_{Setup} in round 2, use $(x_i, \mathsf{com}_{\mathsf{Setup},i}, R^i_{\mathsf{Setup}})$ as the input for P_i and continue running Π^{M} against the adversary $\mathcal{M}_0(\mathcal{A})$.

1. Run the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ against the adversary $\mathcal{M}_0(\mathcal{A})$, $\boxed{\mathcal{M}_1(\mathcal{A})}$, $\mathcal{M}_2(\mathcal{A})$.

For every query $(x_j, \mathsf{com}_{\mathsf{Setup},j}, \mathsf{open}^j_{\mathsf{Setup}}, r^{\mathsf{Enc}}_j)_{j \in I}$ and $\mathsf{com}_{\mathsf{Setup},i}$ issued by $\Pi^{\mathsf{M}}.\mathcal{S}$ do the following:

1. Check that $\mathsf{com}_{\mathsf{Setup},i} = \mathsf{com}_{\mathsf{Setup},j}, R^i_{\mathsf{Setup}} = R^j_{\mathsf{Setup}}$ for all $j \in I$, $\mathsf{com}^{j \to k}_{\mathsf{Setup}} = \mathsf{Com}(r^{j \to k}_{\mathsf{Setup}}; r^{j \to k}_{\mathsf{com}})$ with $\mathsf{open}^j_{\mathsf{Setup}} := (r^{j \to k}_{\mathsf{Setup}}, r^{j \to k}_{\mathsf{com}})_{k \in [n]}$ for all $j \in I, k \in [n]$ and $\tilde{r}^{j \to k}_{\mathsf{Setup}} = r^{j \to k}_{\mathsf{Setup}}$ with $R^i_{\mathsf{Setup}} := (\tilde{r}^{j \to k}_{\mathsf{Setup}})_{j \in [n], k \in [n] \setminus \{j\}}$ for all $j \in I, k \in [n] \setminus \{j\}$. If one of these checks fails, **Abort**.
2. Compute $r^{\mathsf{Setup}}_i = \bigoplus_{k \in [n]} r^{k \to i}_{\mathsf{Setup}}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_i)$.

2. Compute $r^{\mathsf{Setup}}_i = \bigoplus_{k \in [n] \setminus \{i\}} r^{k \to i}_{\mathsf{Setup}} \oplus r^{i \to i}_{\mathsf{Setup}'}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_i)$.

3. Query the ideal functionality using $\{x_j\}_{j \in I}$ and receive $C(x_1, \ldots, x_n)$ as an output.
4. Compute $r^{\mathsf{Setup}}_j = \bigoplus_{k \in [n] \setminus \{i\}} r^{k \to j}_{\mathsf{Setup}} \oplus r^{i \to j}_{\mathsf{Setup}'}$ for all $j \in I$ and generate $\mathsf{msk}_j \leftarrow \mathsf{FE}_j.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_j)$. Simulate a ciphertext $\mathsf{ct} \leftarrow \mathsf{DFEC}.\mathcal{S}(\{\mathsf{msk}_i\}_{i \in [n]}, C, C(x_1, \ldots, x_n), I)$ and send $\mathsf{ct}^* := (\mathsf{ct}, \{r^{k \to l}_{\mathsf{Setup}}\}_{k \in [n] \setminus \{i\}, l \in [n] \setminus \{k\}} \cup \{r^{i \to l}_{\mathsf{Setup}'}\}_{l \in [n] \setminus \{i\}})$ to $\Pi^{\mathsf{M}}.\mathcal{S}$.

3. Compute $r^{\mathsf{Setup}}_j = \bigoplus_{k \in [n] \setminus \{i\}} r^{k \to j}_{\mathsf{Setup}} \oplus r^{i \to j}_{\mathsf{Setup}'}$, generate $\mathsf{msk}_j \leftarrow \mathsf{FE}_j.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_j)$ for all $j \in I$, compute $\mathsf{ct} \leftarrow \mathsf{DFEC}.\mathsf{Enc}(\{\mathsf{msk}_i\}_{i \in [n]}, \{x_i\}_{i \in [n]})$ and send $\mathsf{ct}^* := (\mathsf{ct}, \{r^{k \to l}_{\mathsf{Setup}}\}_{k \in [n] \setminus \{i\}, l \in [n] \setminus \{k\}} \cup \{r^{i \to l}_{\mathsf{Setup}'}\}_{l \in [n] \setminus \{i\}})$ to $\Pi^{\mathsf{M}}.\mathcal{S}$.

3. Compute $r^{\mathsf{Setup}}_j = \bigoplus_{k \in [n]} r^{k \to j}_{\mathsf{Setup}}$, generate $\mathsf{msk}_j \leftarrow \mathsf{FE}_j.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_j)$ for all $j \in I$, compute $\mathsf{ct} \leftarrow \mathsf{DFEC}.\mathsf{Enc}(\{\mathsf{msk}_i\}_{i \in [n]}, \{x_i\}_{i \in [n]})$ and send $\mathsf{ct}^* := (\mathsf{ct}, \{r^{k \to l}_{\mathsf{Setup}}\}_{k \in [n], l \in [n] \setminus \{k\}})$ to $\Pi^{\mathsf{M}}.\mathcal{S}$

When $\Pi^{\mathsf{M}}.\mathcal{S}$ stops, output what $\Pi^{\mathsf{M}}.\mathcal{S}$ outputs.

Fig. 18: Description of the hybrids $\mathsf{H}_0, \ldots, \mathsf{H}_4$, where the machines $\mathcal{M}_0, \ldots, \mathcal{M}_3$ are defined in Fig. 19

$\underline{\mathcal{M}_0(\mathcal{A}, r_{\mathcal{A}}), \boxed{\mathcal{M}_1(\mathcal{A}, r_{\mathcal{A}})}, \boxed{\mathcal{M}_2(\mathcal{A}, r_{\mathcal{A}})}:}$

Initialization Run $\mathcal{A}$ using the randomness $r_{\mathcal{A}}$.

Round 1.

1. Receive the message $\mathsf{msg}_{1,i}$ in the left session.
2. Sample $r^{i \to k}_{\mathsf{Setup}}, r^{i \to k}_{\mathsf{com}}$ for all $k \in [n]$, compute $\mathsf{com}^{i \to k}_{\mathsf{Setup}} = \mathsf{Com}(r^{i \to k}_{\mathsf{Setup}}; r^{i \to k}_{\mathsf{com}})$, set $\mathsf{com}^{i}_{\mathsf{Setup}} := \{\mathsf{com}^{i \to k}_{\mathsf{Setup}}\}_{k \in [n]}$ and $\mathsf{open}^{i}_{\mathsf{Setup}} := (r^{i \to k}_{\mathsf{Setup}}, r^{i \to k}_{\mathsf{com}})_{k \in [n]}$.
3. Send $(\mathsf{msg}_{1,i}, \mathsf{com}^{i}_{\mathsf{Setup}})$ in the right session.
4. Receive $(\mathsf{msg}_{1,j}, \mathsf{com}^{j}_{\mathsf{Setup}})_{j \in I}$ as a reply in the right session and output $\{\mathsf{msg}_{1,j}\}_{j \in I}$ in the left session.

Round 2.

1. Receive the message $\mathsf{msg}_{2,i}$ in the left session.
2. $\boxed{\text{Sample random values } r^{i \to j}_{\mathsf{Setup}'}.}$
3. Output $\mathsf{msg}_{2,i}, \boxed{\{r^{i \to j}_{\mathsf{Setup}'}\}_{j \in I}}, \{r^{i \to j}_{\mathsf{Setup}}\}_{j \in I}$ in the right session.
4. Receive $(\mathsf{msg}_{2,j}, r^{j \to i}_{\mathsf{Setup}})_{j \in I}$ as a reply in the right session and send $\{\mathsf{msg}_{2,j}\}_{j \in I}$ in the left session.

For each round $k \in \{3, \ldots, \ell - 1\}$.

1. Upon receiving the message $\mathsf{msg}_{k,i}$ from the left session forward it to $\mathcal{A}$.
2. Receive the messages $\{\mathsf{msg}_{k,j}\}_{j \in I}$ and forward them in the left session.

Round ℓ.

1. Receive the message $\mathsf{msg}_{\ell,i}$ in the left session.
2. Compute $r^{\mathsf{Setup}}_{i} = \bigoplus_{k \in [n]} r^{k \to i}_{\mathsf{Setup}}$ $\boxed{r^{\mathsf{Setup}}_{i} = \bigoplus_{k \in [n] \setminus \{i\}} r^{k \to i}_{\mathsf{Setup}} \oplus r^{i \to i}_{\mathsf{Setup}'}}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_{i})$, compute the partition of C, i.e. $(C_1 \ldots, C_n) \leftarrow \mathsf{DFEC}.\mathsf{Partition}(1^\lambda, C)$ and generate $\mathsf{sk}_i \leftarrow \mathsf{FE}_i.\mathsf{KeyGen}(\mathsf{msk}_i, C_i; r^{\mathsf{KeyGen}}_{i})$ for a random r^{KeyGen}_{i}.
3. Send $(\mathsf{msg}_{\ell,i}, \mathsf{sk}_i)$ in the right session.
4. Receive $(\mathsf{msg}_{\ell,j})_{j \in I}$ as a reply in the right session and output $\{\mathsf{msg}_{\ell,j}\}_{j \in I}$ in the left session.

Fig. 19: The description of the augmented machines $\mathcal{M}_0, \ldots, \mathcal{M}_3$ for the hybrids $\mathsf{H}_0, \ldots, \mathsf{H}_4$.

Proof. The difference between H_1 and H_2 is that the values sent in the third round are replaced by random values (i.e., they are different from the committed values in the first round). Assuming that the output distribution of H_1 and H_2 are distinguishable, we can construct an adversary $\mathcal{A}'$ that breaks the hiding of Com. The adversary $\mathcal{A}'$ works as follows.

1. Sample two sets of random values $\{r^{i\to j}_{\mathsf{Setup}}\}_{j\in[n]\setminus\{i\}}, \{r^{i\to j}_{\mathsf{Setup}'}\}_{j\in[n]\setminus\{i\}}$ and send them to the challenger $\mathcal{C}$.
2. Upon receiving $\{\mathsf{com}^{i\to j}_{\mathsf{Setup}}\}_{j\in[n]\setminus\{i\}}$ from $\mathcal{C}$ sample the values $r^{i\to i}_{\mathsf{Setup}}, r^{i\to i}_{\mathsf{com}} \leftarrow \{0,1\}^{\lambda}$ and compute $\mathsf{com}^{i\to i}_{\mathsf{Setup}} := \mathsf{Com}(r^{i\to i}_{\mathsf{Setup}}; r^{i\to i}_{\mathsf{com}})$
3. Set $\mathsf{com}^{i}_{\mathsf{Setup}} := \{\mathsf{com}^{i\to j}_{\mathsf{Setup}}\}_{j\in[n]}$.
4. Act exactly as in H_1 (and H_2) with the following differences:
   (a) In round one use $\mathsf{com}^{i}_{\mathsf{Setup}} := \{\mathsf{com}^{i\to j}_{\mathsf{Setup}}\}_{j\in[n]}$ instead of freshly generated commitments
   (b) In round two output the values $\{r^{i\to j}_{\mathsf{Setup}}\}_{j\in[n]\setminus\{i\}}$.
   (c) For every query $(x_j, \mathsf{com}_{\mathsf{Setup},j}, \mathsf{open}^{j}_{\mathsf{Setup}}, R^{j}_{\mathsf{Setup}}, r^{\mathsf{Enc}}_{j})_{j\in I}$ and $\mathsf{com}_{\mathsf{Setup},i}, R^{i}_{\mathsf{Setup}}$ asked by the simulator Π^{M}, check that $\mathsf{com}_{\mathsf{Setup},i} = \mathsf{com}_{\mathsf{Setup},j}, R^{i}_{\mathsf{Setup}} = R^{j}_{\mathsf{Setup}}$ for all $j \in I$, $\mathsf{com}^{j\to k}_{\mathsf{Setup}} = \mathsf{Com}(r^{j\to k}_{\mathsf{Setup}}; r^{j\to k}_{\mathsf{com}})$ with $\mathsf{open}^{j}_{\mathsf{Setup}} := (r^{j\to k}_{\mathsf{Setup}}, r^{j\to k}_{\mathsf{com}})_{k\in[n]}$ for all $j \in I, k \in [n]$ and $\tilde{r}^{j\to k}_{\mathsf{Setup}} = r^{j\to k}_{\mathsf{Setup}}$ with $R^{i}_{\mathsf{Setup}} := (\tilde{r}^{j\to k}_{\mathsf{Setup}})_{j\in[n],k\in[n]\setminus\{j\}}$ for all $j \in I, k \in [n]\setminus\{j\}$. If one of these check fails, reply to the simulator with **Abort**. In the case that these tests pass, compute $r^{\mathsf{Setup}}_{i} = \bigoplus_{k\in[n]} r^{k\to i}_{\mathsf{Setup}}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^{\lambda}; r^{\mathsf{Setup}}_{i})$. Compute $r^{\mathsf{Setup}}_{j} = \bigoplus_{k\in[n]} r^{k\to j}_{\mathsf{Setup}}$, generate $\mathsf{msk}_j \leftarrow \mathsf{FE}_j.\mathsf{Setup}(1^{\lambda}; r^{\mathsf{Setup}}_{j})$ for all $j \in I$, compute $\mathsf{ct} \leftarrow \mathsf{DFEC.Enc}(\{\mathsf{msk}_i\}_{i\in[n]}, \{x_i\}_{i\in[n]})$ and send $\mathsf{ct}^* := (\mathsf{ct}, \{r^{k\to l}_{\mathsf{Setup}}\}_{k\in[n],l\in[n]\setminus\{k\}})$ to $\Pi^{\mathsf{M}}.\mathcal{S}$
   (d) Output what $\mathcal{A}$ outputs.

The proof ends with the observation that the output of $\mathcal{A}''$ in the case where $\mathcal{C}$ has computed the commitments using the values $\{r^{i\to j}_{\mathsf{Setup}}\}_{j\in[n]\setminus\{i\}}$ corresponds to the output of H_1, and to the output of H_2 otherwise. □

Claim 3 (Transition from H_2 to H_3) *If* Com *is a computationally hiding commitment scheme, then the output distribution of the hybrids* H_2 *and* H_3 *are computationally indistinguishable.*

Proof. In the transition from hybrid H_2 to hybrid H_3 the randomness of the party P_i that is used to generate the master secret key msk_i changes from being determined by the committed values of all the parties, i.e. $r^{\mathsf{Setup}}_{i} = \bigoplus_{k\in[n]} r^{k\to i}_{\mathsf{Setup}}$ to a random value that is independent of the commitments $\{\mathsf{com}^{k\to i}_{\mathsf{Setup}}\}_{k\in[n]}$ by computing $r^{\mathsf{Setup}}_{i} = \bigoplus_{k\in[n]\setminus\{i\}} r^{k\to i}_{\mathsf{Setup}} \oplus r^{i\to i}_{\mathsf{Setup}'}$ with a random value $r^{i\to i}_{\mathsf{Setup}'}$.

Assuming that the output distribution of H_1 and H_2 are distinguishable, we can construct an adversary $\mathcal{A}'$ that breaks the hiding of Com. The adversary $\mathcal{A}'$ works as follows.

1. Sample two random values $r^{i\to i}_{\mathsf{Setup}}, r^{i\to i}_{\mathsf{Setup}'}$ and send them to the challenger $\mathcal{C}$.
2. Upon receiving $\mathsf{com}^{i\to i}_{\mathsf{Setup}}$ from $\mathcal{C}$ act exactly as in H_2 (and H_3) with the following differences:
   (a) In round 1 use $\mathsf{com}^{i\to i}_{\mathsf{Setup}}$ instead of freshly generated commitments.
   (b) In round ℓ compute $r^{\mathsf{Setup}}_{i} = \bigoplus_{k\in[n]} r^{k\to i}_{\mathsf{Setup}}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^{\lambda}; r^{\mathsf{Setup}}_{i})$, compute the partition of C, i.e. $(C_1 \ldots, C_n) \leftarrow \mathsf{DFEC.Partition}(1^{\lambda}, C)$ and generate $\mathsf{sk}_i \leftarrow \mathsf{FE}_i.\mathsf{KeyGen}(\mathsf{msk}_i, C_i; r^{\mathsf{KeyGen}}_{i})$ for a random r^{KeyGen}_{i}.
   (c) For every query $(x_j, \mathsf{com}_{\mathsf{Setup},j}, \mathsf{open}^{j}_{\mathsf{Setup}}, R^{j}_{\mathsf{Setup}}, r^{\mathsf{Enc}}_{j})_{j\in I}$ and $\mathsf{com}_{\mathsf{Setup},i}, R^{i}_{\mathsf{Setup}}$ asked by the simulator Π^{M}, check that $\mathsf{com}_{\mathsf{Setup},i} = \mathsf{com}_{\mathsf{Setup},j}, R^{i}_{\mathsf{Setup}} = R^{j}_{\mathsf{Setup}}$ for all $j \in I$, $\mathsf{com}^{j\to k}_{\mathsf{Setup}} = \mathsf{Com}(r^{j\to k}_{\mathsf{Setup}}; r^{j\to k}_{\mathsf{com}})$ with $\mathsf{open}^{j}_{\mathsf{Setup}} := (r^{j\to k}_{\mathsf{Setup}}, r^{j\to k}_{\mathsf{com}})_{k\in[n]}$ for all $j \in I, k \in [n]$ and $\tilde{r}^{j\to k}_{\mathsf{Setup}} = r^{j\to k}_{\mathsf{Setup}}$ with $R^{i}_{\mathsf{Setup}} := (\tilde{r}^{j\to k}_{\mathsf{Setup}})_{j\in[n],k\in[n]\setminus\{j\}}$ for all $j \in I, k \in [n]\setminus\{j\}$. If one of these check fails, reply to the simulator with **Abort**. In the case that these

tests pass, compute $r_i^{\mathsf{Setup}} = \bigoplus_{k \in [n]} r_{\mathsf{Setup}}^{k \to i}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$. Compute $r_j^{\mathsf{Setup}} = \bigoplus_{k \in [n]} r_{\mathsf{Setup}}^{k \to j}$, generate $\mathsf{msk}_j \leftarrow \mathsf{FE}_j.\mathsf{Setup}(1^\lambda; r_j^{\mathsf{Setup}})$ for all $j \in I$, compute $\mathsf{ct} \leftarrow \mathsf{DFEC.Enc}(\{\mathsf{msk}_i\}_{i \in [n]}, \{x_i\}_{i \in [n]})$ and send $\mathsf{ct}^* := (\mathsf{ct}, \{r_{\mathsf{Setup}}^{k \to l}\}_{k \in [n], l \in [n] \setminus \{k\}})$ to $\Pi^{\mathsf{M}}.\mathcal{S}$.

3. Output what $\mathcal{A}$ outputs.

The proof ends with the observation that the output of $\mathcal{A}''$ in the case where $\mathcal{C}$ computes the commitments using the value $r_{\mathsf{Setup}}^{i \to i}$ corresponds to the output of H_2, and to the output of H_3 otherwise. □

To make the next transition possible, we need to introduce an intermediate hybrid $\mathsf{H}_3^\star$. This hybrid works as H_3 with the difference that once the adversary $\mathcal{A}$ has received the third round, it is rewound up to the end of the first round after $\mathcal{A}$ has provided its output. In the next step, a set of new random values $\{r_{\mathsf{Setup}''}^{i \to j}\}_{j \in [n] \setminus \{i\}}$ is sampled and used to complete the interaction with the adversary $\mathcal{A}$ instead of $\{r_{\mathsf{Setup}'}^{i \to j}\}_{j \in [n] \setminus \{i\}}$ accordingly to H_3. We introduce this hybrid to simplify our last reduction to the security of the FE combiner. We provide a more detailed description of the hybrid $\mathsf{H}_3^\star$ below.

Hybrid $\mathsf{H}_3^\star$: This hybrid works as hybrid H_3 with the difference that look ahead threads are created. In more detail, in this hybrid, the first three rounds of the protocol are executed as in hybrid H_3. When the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ of the inner MPC sends the query $(x_j, \mathsf{com}_{\mathsf{Setup},j}, \mathsf{open}_{\mathsf{Setup}}^j, R_{\mathsf{Setup}}^i, r_j^{\mathsf{Enc}})_{j \in I}$, the simulator checks that $\mathsf{com}_{\mathsf{Setup},i} = \mathsf{com}_{\mathsf{Setup},j}$, $R_{\mathsf{Setup}}^i = R_{\mathsf{Setup}}^j$ for all $j \in I$, $\mathsf{com}_{\mathsf{Setup}}^{j \to k} = \mathsf{Com}(r_{\mathsf{Setup}}^{j \to k}; r_{\mathsf{com}}^{j \to k})$ with $\mathsf{open}_{\mathsf{Setup}}^j := (r_{\mathsf{Setup}}^{j \to k}, r_{\mathsf{com}}^{j \to k})_{k \in [n]}$ for all $j \in I, k \in [n]$ and $\tilde{r}_{\mathsf{Setup}}^{j \to k} = r_{\mathsf{Setup}}^{j \to k}$ with $R_{\mathsf{Setup}}^i := (\tilde{r}_{\mathsf{Setup}}^{j \to k})_{j \in [n], k \in [n] \setminus \{j\}}$ for all $j \in I, k \in [n] \setminus \{j\}$.
We now distinguish between two cases 1) the above check fails and 2) the above check is successful. We denote the event in which 1) occurs with E_1 and the event in which 2) occurs with E_2. In the case that E_1 occurs, the hybrid behaves exactly as H_3 (i.e., no rewinds are needed) and the simulator $\mathcal{S}$ replies to the query $(x_j, \mathsf{com}_{\mathsf{Setup},j}, \mathsf{open}_{\mathsf{Setup}}^j, R_{\mathsf{Setup}}^i, r_j^{\mathsf{Enc}})_{j \in I}$ with **Abort.**. In the case of E_2, the current thread becomes a look ahead thread and the execution is rewound until the beginning of the second round. Then the second and third round is executed again with the difference that the random values $\{r_{\mathsf{Setup}'}^{i \to j}\}_{j \in I}$ sent in the second round of the look ahead thread are replaced with freshly generated random values $\{r_{\mathsf{Setup}''}^{i \to j}\}_{j \in I}$.
We distinguish again between the two cases where 1) the query made by $\Pi^{\mathsf{M}}.\mathcal{S}$ to its ideal functionality is valid (i.e., the ideal functionality for C_{ct} does not return **Abort**) or 2) the query made by $\Pi^{\mathsf{M}}.\mathcal{S}$ is invalid and the ideal functionality returns **Abort**[17]. We denote with F_1 the event that the first case occurs and with F_2 the event that the second case occurs. In the case of F_2, we rewind the adversary again, create another look ahead thread and behave as described before. We repeat this procedure until the ideal functionality query asked by the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ fulfills the condition, which puts us into event F_1. Since we need to ensure that our hybrid experiment runs in polynomial time, we need to argue that the adversary $\mathcal{A}$ gets rewound at most a polynomial number of times. We recall that in this proof we are assuming that the event E_1 happens with some non-negligbile probability p. Given that the view of the adversary during the look-ahead threads is the same as in the main thread, we can claim that $\Pr[E_1] = \Pr[F_1]$. Hence, the number of rewinds is polynomial in the security parameter. Given that $\Pr[E_1] = \Pr[F_1]$ and that the view of the adversary in

[17] We recall that the ideal functionality for C_{ct} is emulated by the simulator.

the look-ahead threads and in the main thread is identical, we can claim that the output distribution of H_3 and $\mathsf{H}_3^\star$ are identical.

Claim 4 (Transition from $\mathsf{H}_3^\star$ to $\mathsf{H}_4^\star$) *If* DFEC *is a single-key simulation secure decomposable FE combiner, then the hybrids* $\mathsf{H}_3^\star$ *and* $\mathsf{H}_4^\star$ *are computationally indistinguishable.*

Proof. The hybrids $\mathsf{H}_3^\star$ and $\mathsf{H}_4^\star$ differ in the generation of the ciphertext ct that is output by the inner MPC protocol $\mathit{\Pi}^{\mathsf{M}}$. In hybrid $\mathsf{H}_3^\star$ the ciphertext is generated by encrypting the messages $(x_1, \ldots, x_n)$ using the encryption procedure $\mathsf{DFEC.Enc}$, whereas in hybrid $\mathsf{H}_4^\star$ the ciphertext is simulated using the inputs $(\{x_j\}_{j \in I}, C, C(x_1, \ldots, x_n), I)$ and the simulator of the functional encryption combiner $\mathsf{DFEC}.\mathcal{S}$.

Assuming that the output distribution of $\mathsf{H}_3^\star$ and $\mathsf{H}_4^\star$ are distinguishable, we can construct an adversary $\mathcal{A}'$ that breaks the single-key simulation security of the decomposable FE combiner DFEC. The adversary $\mathcal{A}'$ works as follows.

1. Interact with $\mathcal{A}$ accordingly to $\mathsf{H}_3^\star$ until the look ahead threads are created.
2. In the case that event E_1 occurs, the reduction stops here and outputs **Abort**. In the case that E_2 occurs save the values $r_{\mathsf{Setup}}^{j \to k}$ for all $j \in I, k \in [n]$ and submit the set of corrupted parties I and the circuit C to its underlying challenger $\mathcal{C}$.
3. Upon receiving the master secret keys $\{\mathsf{msk}_j\}_{j \in I}$, as well as the functional key $\mathsf{sk}_C = (\mathsf{sk}_{C_1}, \ldots, \mathsf{sk}_{C_n})$ with $(C_1, \ldots, C_n) \leftarrow \mathsf{DFEC.Partition}(1^\lambda, C)$ for the circuit C compute $r_{\mathsf{Setup}''}^{i \to j} = \bigoplus_{k \in [n] \setminus \{i\}} r_{\mathsf{Setup}}^{k \to j} \oplus r_j^{\mathsf{Setup}}$ for all $j \in I$ with $\{r_{\mathsf{Setup}}^{k \to j}\}_{j \in I, k \in [n]}$ learned from the creation of the look-ahead threads.
4. Return to the main thread and output $(r_{\mathsf{Setup}''}^{i \to j})_{j \in [n] \setminus \{i\}}$ in the second round. Continue with the execution until event F_1 occurs.
5. Act exactly as in $\mathsf{H}_3^\star$ (and $\mathsf{H}_4^\star$) until the commitments of a query $(x_j, \mathsf{com}_{\mathsf{Setup},j}, \mathsf{open}_{\mathsf{Setup}}^j, R_{\mathsf{Setup}}^j, r_j^{\mathsf{Enc}})_{j \in I}$ and $\mathsf{com}_{\mathsf{Setup},i}, R_{\mathsf{Setup}}^i$, asked by the simulator of $\mathit{\Pi}^{\mathsf{M}}$, are checked for equality and correctness.
6. Query the ideal functionality using $\{x_j\}_{j \in I}$ and receive $C(x_1, \ldots, x_n)$ as an answer.
7. Submit $(\{x_k\}_{k \in [n]}, C(x_1, \ldots, x_n))$ to the challenger $\mathcal{C}$.
8. Upon receiving ct, send $\mathsf{ct}^* := (\mathsf{ct}, \{r_{\mathsf{Setup}}^{k \to l}\}_{k \in [n] \setminus \{i\}, l \in [n] \setminus \{k\}}, \{r_{\mathsf{Setup}''}^{i \to l}\}_{l \in [n] \setminus \{i\}})$ to $\mathit{\Pi}^{\mathsf{M}}.\mathcal{S}$.
9. Output what $\mathcal{A}$ outputs.

The proof ends with the observation that the output of $\mathcal{A}''$ in the case where $\mathcal{C}$ computes the ciphertext using the values $\{x_k\}_{k \in [n]}$ corresponds to the output of $\mathsf{H}_3^\star$, and to the output of $\mathsf{H}_4^\star$ otherwise.

For the final transition from $\mathsf{H}_4^\star$ to H_4, we need to simulate the ciphertext ct without the creation of look ahead threads. Since we do not need to rely on the challenger of the functional encryption combiner DFEC to simulate the ciphertext ct in H_4, it is also not necessary to program the master secret keys of the corrupted parties $\{\mathsf{msk}_j\}_{j \in I}$ to match the master secret keys generated by the challenger. This allows the simulator $\mathcal{S}$ of the MPC protocol $\mathit{\Pi}^{\mathsf{FE}}$ to simulate the ciphertext using the master secret keys $\{\mathsf{msk}_j\}_{j \in I}$ defined by the randomness $\{r_j^{\mathsf{Setup}}\}$, i.e $\mathsf{msk}_j \leftarrow \mathsf{FE.Setup}(1^\lambda; r_j^{\mathsf{Setup}})$ for all $j \in I$ with $r_j^{\mathsf{Setup}} = \bigoplus_{k \in [n] \setminus \{i\}} r_{\mathsf{Setup}}^{k \to j} \oplus r_{\mathsf{Setup}'}^{i \to j}$. Therefore there is no need to sample the additional random values $\{r_{\mathsf{Setup}''}^{i \to l}\}_{l \in [n] \setminus \{i\}}$.

These changes do not affect the output distribution of $\mathsf{H}_4^\star$ and H_4, which makes the two hybrids perfectly indistinguishable. □

The following theorem follows immediately from Theorem 2 and the definition of strong succinct FE combiners.

Theorem 3. *Let* DFEC *be a succinct single-key simulation secure decomposable FE combiner, then* Π^{FE} *is a circuit-scalable secure MPC protocol that realizes any single-output functionality with knowledge of outputs.*

Instantiations. To instantiate our compiler we need an ℓ-round k-Signaling MPC protocol and a succinct decomposable FE combiner. From Theorem 1 and from the fact that the 4-round protocols proposed in [BGJ+18, CCG+19] do not require the input to compute the first two rounds, we can construct a k-Signaling MPC protocol assuming that any of these assumptions holds: DDH, QR, N^{th} Residuosity, or existence of malicious-secure OT assumptions holds.

Regarding the FE combiner, in [ABJ+19, Section 4], the authors mention that a functional encryption scheme that fulfills succinctness and can be used as an instantiation for the FE candidates of the the FE combiner is the scheme of Goldwasser et al. [GKP+13]. However, our definition of *strong* succinctness for FE combiner requires the complexity of the setup algorithm to be dependent only on the circuit depth, the input and the output size of the function being computed. For their instantiation, the authors of [ABJ+19] consider the FE protocol protocol proposed in [GKP+13]. This can be instantiated from an attribute based encryption (ABE) scheme and leveled fully-homomorphic encryption (FHE) scheme, and becomes succinct (in the key-size and the description of the encryption circuit) when instantiated with one of the ABE schemes proposed in [BGG+14, GVW15] and one of the leveled FHE schemes of [BGV12, GSW13]. Fortunately, the scheme of Goldwasser et al. [GKP+13] provides succinctness also in the description of the setup algorithm when instantiated with the above ABE and FHE schemes. In more detail, the FE protocol proposed in [GKP+13] runs the setup algorithm of an ABE scheme N times, where $N := \mathrm{poly}(\lambda, d, L_{\mathsf{in}})$. The ABE schemes are used to compute the *evaluation function* Eval for the underling FHE scheme Eval and has depth d (but its size depends on f). Hence, we just need to make sure that also the description of the circuit of the setup procedure of the ABE schemes proposed in [BGG+14, GVW15] depends only on the circuit depth, input and output size of Eval. In the work of Boneh et al. [BGG+14, Section 4] the authors present an attribute based encryption scheme based on the LWE assumption, where the setup algorithm takes as an input a unary representation of the security parameter λ and the input length L_{in} of the circuit that needs to be computed. Therefore the running time of this algorithm only depends on the security parameter and the circuit input-length, which results in $\mathrm{poly}(\lambda, L_{\mathsf{in}})$. The predicate encryption scheme presented in [GVW15, Section 4] is also based on the LWE assumption and the setup algorithm of this construction takes as an input a unary representation of the security parameter λ, the input length L_{in} as well as the depth of the circuit d. This allows us to bound the circuit size of the setup algorithm as $\mathrm{poly}(\lambda, d, L_{\mathsf{in}})$. Hence, we get that the description of the setup circuit of the FE protocol of [GKP+13] is at most $cs_{\mathsf{Setup}} = \mathrm{poly}(\lambda, d, L_{\mathsf{in}}, n)$. Given the above, we can now claim the following corollary.

Theorem 4. *If the LWE assumption holds and any of the DDH, QR,* N^{th} *Residuosity, or existence of malicious-secure OT assumptions holds, then there exists a round optimal (4-round) circuit-scalable MPC protocol that realizes any single-output functionality with knowledge of outputs.*

By relying on the compiler that amplifies security from privacy with knowledge of outputs to full security, based on signatures (Section 6) and the compiler that realizes any functionality from single-output functionalities, based on symmetric encryption (Section 7).

Using the following observations:

1. A signature scheme can be instantiated from OWFs [Rom90];
2. The encryption scheme can be instantiated information-theoretically (just using one-time pad).

we obtain the following corollary.

Corollary 1. *If the LWE assumption holds and any of the DDH, QR, N^{th} Residuosity holds, or there exists a malicious-secure OT, then there exists a round optimal (4-round)* circuit-scalable *MPC protocol that realizes any functionality.*

5 Our Compiler: Circuit-Independent Efficient MPC

In this section we prove our main theorems on how to construct a communication efficient MPC protocol that realizes any functionality f with knowledge of outputs. We refer to Section 2 for a simplified description of the protocol for the two-party case and to Fig. 24 for the formal description of our compiler.

Our Construction makes use of the following cryptographic tools:

- An ℓ-round k-signaling MPC protocol $\Pi^{\mathsf{M}} = (\Pi^{\mathsf{M}}.\mathsf{Round}_1, \dots, \Pi^{\mathsf{M}}.\mathsf{Round}_\ell, \Pi^{\mathsf{M}}.\mathsf{Out})$ (not necessarily communication efficient) with $k \geq 2$ that securely evaluates the function C_{Dec} (described in Fig. 25), where $\Pi^{\mathsf{M}}.\mathsf{Round}_k$ takes the input of the party P_i that we denote with y_i.[18] In the description of our compiler we assume, without loss of generality, that Π^{M} is 2-signaling.[19]
- A multi-key fully homomorphic encryption scheme $\mathsf{MFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ for n keys.

Theorem 5. *Let* MFHE *be a multi-key fully homomorphic encryption scheme with circuit size* cs_{Setup} *for the setup algorithm* $\mathsf{MFHE.Setup}$*, circuit size* cs_{Enc} *for the encryption algorithm* $\mathsf{MFHE.Enc}$*, circuit size* cs_{Dec} *for the decryption algorithm* $\mathsf{MFHE.Dec}$ *and ciphertext size* s_{ct}*, let* Π^{M} *be an* k*-signaling* ℓ*-round MPC protocol that securely realizes* C_{Dec}*, then* Π^{FHE} *is an* ℓ*-round MPC protocol that realizes the single-output functionality* C *with knowledge of outputs which has communication complexity* $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, cs_{\mathsf{Dec}}, s_{\mathsf{ct}})$*.*

We split this theorem into two Lemmas and prove them separately:

Lemma 3. *Let* MFHE *be a multi-key fully homomorphic encryption scheme with circuit size* cs_{Setup} *for the setup algorithm* $\mathsf{MFHE.Setup}$*, circuit size* cs_{Enc} *for the encryption algorithm* $\mathsf{MFHE.Enc}$*, circuit size* cs_{Dec} *for the decryption algorithm* $\mathsf{MFHE.Dec}$ *and ciphertext size* s_{ct}*, then* Π^{FHE} *has communication complexity* $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, cs_{\mathsf{Dec}}, s_{\mathsf{ct}})$*.*

Proof. We divide the analysis of the communication complexity into two steps. In the first step, we analyze the communication complexity of the inner MPC protocol Π^{M} and in the second step the communication complexity of the additional values sent in the outer MPC protocol.

We remark that the operations executed inside the MPC protocol Π^{M}, besides the generation of the public and secret keys, the encryptions and the decryption, have communication complexity $\mathrm{poly}(\lambda, n)$. Since the circuit size of the setup algorithm is cs_{Setup}, the circuit size of the encryption algorithm is cs_{Enc} and the circuit size of the decryption algorithm is cs_{Dec} the resulting message length is $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, cs_{\mathsf{Dec}})$, i.e. $|\mathsf{msg}_{k,i}| = \mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, cs_{\mathsf{Dec}})$ for all $k \in \{1, \dots, \ell\}$ and all $i \in [n]$.

After analyzing the communication complexity of the inner MPC protocol Π^{M}, we continue with the analysis of the messages that are sent in addition to the messages of Π^{M}.

[18] To simplify the description of the protocol we assume that the entire input (the signaled part and the private part) is provided in round k.

[19] It is easy to see that any k'-signaling MPC with $k' > 2$ can be turned into a 2-signaling MPC protocol since that the signaled input received in round 2 can be ignore up to round $k' - 1$.

The additional messages, public keys and ciphertexts, that are output by every party P_i for all $i \in [n]$ in round 1 are of length $\mathrm{poly}(\lambda, n, s_{\mathsf{ct}}, s_{\mathsf{pk}})$. This results in a communication complexity of $\mathrm{poly}(\lambda, n, s_{\mathsf{sk}})$ for the additional messages.

Combining the two analysis yields an overall communication complexity of $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, cs_{\mathsf{Dec}}, s_{\mathsf{ct}}, s_{\mathsf{pk}}, s_{\mathsf{sk}})$. □

To specify the values $cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, cs_{\mathsf{Dec}}$ and s_{sk} we apply the definition of compactness for MFHE (Definition 10), which results in $s_{\mathsf{ct}} = \mathrm{poly}(\lambda, n)$. For the circuit sizes $cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}$ and cs_{Dec} of the setup algorithm Setup, the encryption algorithm Enc and the decryption algorithm Dec, it holds that $cs_{\mathsf{Setup}} = \mathrm{poly}(\lambda)$, $cs_{\mathsf{Enc}} = \mathrm{poly}(\lambda, L_{\mathsf{in}})$ and $cs_{\mathsf{Dec}} = \mathrm{poly}(\lambda, n, L_{\mathsf{out}})$. This results in an overall communication complexity of $\mathrm{poly}(\lambda, n, L_{\mathsf{in}}, L_{\mathsf{out}})$.

Lemma 4. *Let* MFHE *be a multi-key fully homomorphic encryption scheme, and let* Π^{M} *be a* k*-signaling* ℓ*-round MPC protocol (with* $k \geq 2$*) that securely realizes* C_{Dec}*, then* Π^{FHE} *is an* ℓ*-round MPC protocol that realizes the single-output functionality* C*.*

Proof. To prove our lemma we need to show that for every PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $I \subset [n]$ the following quantity is negligible:

$$|\Pr[\mathrm{Real}_{\Pi^{\mathsf{FHE}},\mathcal{A}(z),I}(\lambda, \boldsymbol{x}) = 1] - \Pr[\mathrm{Ideal}^{\mathsf{PKO}}_{C,\mathcal{S}(z),I}(\lambda, \boldsymbol{x}) = 1]| \ ,$$

where $\boldsymbol{x} = \{x_i\}_{i \in [n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$.

As before, simplicity, we assume that all but one of the parties are corrupted. We denote the set that contains the indices of all the corrupted parties as I, i.e. $|I| = n - 1$. Before describing how $\mathcal{S}$ works, we define an algorithm $\mathcal{M}$ that we refer to as the *augmented machine*. The augmented machine internally runs the adversary $\mathcal{A}$ (we refer to this as the right session), and acts as a proxy between $\mathcal{A}$ and its external interface with respect to the messages of Π^{M} (we refer to this as the left session). To describe our simulator we then need to described the augmented machine $\mathcal{M}$ and how $\mathcal{S}$ interacts with it (i.e., how the messages of Π^{M} are computed).

The reason why we describe our simulator via the augmented machine $\mathcal{M}$ is to deal with the rewinds that the simulator of Π^{M} might do. We note that $\mathcal{M}$ acts as an adversary for the protocol Π^{M}. Hence, we can consider the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ for Π^{M} (which exists by assumption) for the adversary $\mathcal{M}$. Our simulator $\mathcal{S}$ will then simply run $\Pi^{\mathsf{M}}.\mathcal{S}$. For the formal description of $\mathcal{M}$ we refer to Fig. 20 and for the formal description of $\mathcal{S}$ we refer to Fig. 21.

```
$\mathcal{M}(r_{\mathcal{A}})$ :

Round 1.
    1. Receive the message $\mathsf{msg}_{1,i}$ in the left session.
    2. Sample $r_i^{\mathsf{Setup}} \leftarrow \{0,1\}^\lambda$ and compute $(\mathsf{pk}_i, \mathsf{sk}_i) := \mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$.
    3. Sample $r_i^{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$ and compute $\mathsf{ct}_i := \mathsf{Enc}(\mathsf{pk}_i, 0; r_i^{\mathsf{Enc}})$.
    4. Output $(\mathsf{msg}_{1,i}, \mathsf{pk}_i, \mathsf{ct}_i)$ in the right session.
    5. Receive $(\mathsf{msg}_{1,j}, \mathsf{pk}^i_j, \mathsf{ct}^i_j)_{j \in I}$ as a reply in the right session.
    6. Output $\{\mathsf{msg}_{1,j}\}_{j \in I}$ in the left session.
For each round $k \in \{2, \ldots, \ell - 1\}$.
    1. Upon receiving the message $\mathsf{msg}_{k,i}$ from the left session forward it to $\mathcal{A}$.
    2. Receive the messages $\{\mathsf{msg}_{k,j}\}_{j \in I}$ and forward them in the left session.
Round $\ell$.
    1. Receive the message $\mathsf{msg}_{\ell,i}$ in the left session.
    2. Send $\mathsf{msg}_{\ell,i}$ in the right session.
    3. Receive $\{\mathsf{msg}_{\ell,j}\}_{j \in I}$ as a reply in the right session and output $\{\mathsf{msg}_{\ell,j}\}_{j \in I}$ in the left session.
```

Fig. 20: The augmented machine $\mathcal{M}$ which emulates the adversary for Π^{M}.

$\underline{\mathcal{S}}$

- Sample a sufficiently long random R and run the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ for the adversary $\mathcal{M}$.
- For every query $(x_j, \mathsf{ct}^j, K^j, \mathsf{sk}_j, r_j^{\mathsf{Setup}}, r_j^{\mathsf{Enc}})_{j \in I}$ issued by $\Pi^{\mathsf{M}}.\mathcal{S}$ do the following:
  1. Compute $\mathsf{ct} := \mathsf{Eval}(C, (\mathsf{pk}_1^i, \mathsf{ct}_1^i), \ldots, (\mathsf{pk}_n^i, \mathsf{ct}_n^i))$.
  2. For all $j \in I$, parse K^j as $(\mathsf{pk}_l^j, \mathsf{ct}_l^j)_{l \in [n]}$.
  3. For all $j \in I, l \in [n]$ check that $(\mathsf{pk}_l^i, \mathsf{ct}_l^i) = (\mathsf{pk}_l^j, \mathsf{ct}_l^j)$.
  4. For all $j \in I$ check that $(\mathsf{pk}_j^j, \mathsf{sk}_j^j) := \mathsf{Setup}(1^\lambda; r_j^{\mathsf{Setup}})$ and $\mathsf{ct}_j^j := \mathsf{Enc}(\mathsf{pk}_j, x_j; r_j^{\mathsf{Enc}})$.
  5. Compute $y := \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{ct})$, $y^j := \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{ct}^j)$ and check that $y = y^j$ for all $j \in I$.
  6. If one of the above checks fails then output $\bot$, continue as follows otherwise.
  7. Query the ideal functionality C using $\{x_j\}_{j \in I}$ and receive $y := C(x_1, \ldots, x_n)$ as an output.
  8. Send y to $\Pi^{\mathsf{M}}.\mathcal{S}$.
- When $\Pi^{\mathsf{M}}.\mathcal{S}$ stops, output what $\Pi^{\mathsf{M}}.\mathcal{S}$ outputs.

Fig. 21: The simulator $\mathcal{S}$ for our protocol Π^{FHE}.

We describe the hybrid experiments that we use to prove the lemma. We first give an informal description:

Hybrid H_0: Hybrid H_0 is identical to the real world.

Hybrid H_1: In hybrid H_1, the messages of the inner MPC protocol Π^{M} are simulated using the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ (which exists by assumption). The transition between hybrid H_0 and H_1 is justified by the malicious security of the MPC protocol Π^{M} and formally proven in Claim 5.

Hybrid H_2: Hybrid H_2 is identical to the ideal world. In this hybrid, the honestly generated ciphertext ct_i is replaced by an encryption of 0 under the same public key instead of an encryption of x_i. The transition between H_1 and H_2 is justified by the IND-CPA security of the multi-key fully homomorphic encryption scheme MFHE and formally proven in Claim 6.

For the formal description of the hybrids, we also use an augmented machine. More precisely, we define a different augmented machine for each hybrid experiment. In addition, for each hybrid experiment H_k with $k \in \{0, 1, 2\}$ we need to specify how to construct the answers to the query made by the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$. For the formal description of the augmented machines we refer to Fig. 23, whereas the formal description of the hybrid experiments is provided in Fig. 22.

H_0, $\boxed{\mathsf{H}_1, \boxed{\mathsf{H}_2}}$

1. Run Π^{M} against the adversary $\mathcal{M}_0(\mathcal{A})$. Upon receiving $\mathsf{ct}^i, K^i, \mathsf{sk}_i, r_i^{\mathsf{Setup}}, r_i^{\mathsf{Enc}}$ in round 2, use $(x_i, \mathsf{ct}^i, K^i, \mathsf{sk}_i, r_i^{\mathsf{Setup}}, r_i^{\mathsf{Enc}})$ as the input for P_i and continue running Π^{M} against the adversary $\mathcal{M}_0(\mathcal{A})$.

1. Run the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$ against the adversary $\mathcal{M}_0(\mathcal{A})$, $\boxed{\mathcal{M}_1(\mathcal{A})}$.

For every query $(x_j, \mathsf{ct}^j, K^j, \mathsf{sk}_j, r_j^{\mathsf{Setup}}, r_j^{\mathsf{Enc}})_{j \in I}$ issued by $\Pi^{\mathsf{M}}.\mathcal{S}$ do the following:

1. Check that $(\mathsf{pk}_j^j, \mathsf{sk}_j^j) := \mathsf{Setup}(1^\lambda; r_j^{\mathsf{Setup}}), \mathsf{ct}_j^j := \mathsf{Enc}(\mathsf{pk}_j, x_j; r_j^{\mathsf{Enc}})$ for all $j \in I$ with $K^j = (\mathsf{pk}_l^j, \mathsf{ct}_l^j)_{l \in [n]}$ and check that $(\mathsf{pk}_l^i, \mathsf{ct}_l^i) = (\mathsf{pk}_l^j, \mathsf{ct}_l^j)$ for all $j \in I, l \in [n]$. Compute $\mathsf{ct} := \mathsf{Eval}(C, (\mathsf{pk}_1^i, \mathsf{ct}_1^i), \ldots, (\mathsf{pk}_n^i, \mathsf{ct}_n^i)), y := \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{ct}), y^j := \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{ct}^j)$ and check that $y = y^j$ for all $j \in I$. If one of these checks fails, **Abort**.

3. Query the ideal functionality C using $\{x_j\}_{j \in I}$, receive $y := C(x_1, \ldots, x_n)$ as an output and send y to $\Pi^{\mathsf{M}}.\mathcal{S}$.

3. Send y to $\Pi^{\mathsf{M}}.\mathcal{S}$

When $\Pi^{\mathsf{M}}.\mathcal{S}$ stops, output what $\Pi^{\mathsf{M}}.\mathcal{S}$ outputs.

Fig. 22: Description of the hybrids $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$, where the machines $\mathcal{M}_0$ and $\mathcal{M}_1$ are defined in Fig. 23

$\mathcal{M}_0(r_{\mathcal{A}})$, $\boxed{\mathcal{M}_1(\mathcal{A}, r_{\mathcal{A}})}$:

Round 1.

1. Receive the message $\mathsf{msg}_{1,i}$ in the left session.
2. Sample $r_i^{\mathsf{Setup}} \leftarrow \{0,1\}^\lambda$ and compute $(\mathsf{pk}_i, \mathsf{sk}_i) := \mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$.
3. Sample $r_i^{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$ and compute $\mathsf{ct}_i := \mathsf{Enc}(\mathsf{pk}_i, x_i; r_i^{\mathsf{Enc}})$ $\boxed{\mathsf{ct}_i := \mathsf{Enc}(\mathsf{pk}_i, 0; r_i^{\mathsf{Enc}})}$.
4. Output $(\mathsf{msg}_{1,i}, \mathsf{pk}_i, \mathsf{ct}_i)$ in the right session.
5. Receive $(\mathsf{msg}_{1,j}, \mathsf{pk}_j^i, \mathsf{ct}_j^i)_{j \in I}$ as a reply in the right session.
6. Output $\{\mathsf{msg}_{1,j}\}_{j \in I}$ in the left session.

For each round $k \in \{2, \ldots, \ell - 1\}$**.**

1. Upon receiving the message $\mathsf{msg}_{k,i}$ from the left session forward it to $\mathcal{A}$.
2. Receive the messages $\{\mathsf{msg}_{k,j}\}_{j \in I}$ and forward them in the left session.

Round ℓ**.**

1. Receive the message $\mathsf{msg}_{\ell,i}$ in the left session.
2. Send $\mathsf{msg}_{\ell,i}$ in the right session.
3. Receive $\{\mathsf{msg}_{\ell,j}\}_{j \in I}$ as a reply in the right session and output $\{\mathsf{msg}_{\ell,j}\}_{j \in I}$ in the left session.

Fig. 23: The augmented machine $\mathcal{M}$ which emulates the adversary for Π^{M}.

□

Claim 5 (Transition from H_0 to H_1) *Let Π^{M} be a maliciously secure MPC protocol, then the output distributions of the hybrid experiments H_0 and H_1 are computationally indistinguishable.*

Proof. By assumption we know that for every PPT adversary $\mathcal{A}'$ there exists a PPT adversary $\mathcal{S}'$ such that for any $I \subset [n]$ the following quantity is negligible:

$$|\Pr[\mathrm{Real}_{\Pi^{\mathsf{M}}, \mathcal{A}(z), I}(k, \boldsymbol{x}) = 1] - \Pr[\mathrm{Ideal}_{C_{\mathsf{Dec}}, \mathcal{S}'(z), I}(k, \boldsymbol{x}) = 1]|$$

where $\boldsymbol{x} = \{x_i\}_{i\in[n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$.

Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids with non-negligible probability then we can use the adversary $\mathcal{A}' := \mathcal{M}_0(\mathcal{A}, \cdot)$ to break the security of Π^{M}. The description of the augmented machine $\mathcal{M}_0$ can be found in Fig. 23.

Note that $\mathcal{M}_0(\mathcal{A}, \cdot)$ is a valid adversary for Π^{M} as, in each round $k \in [\ell]$ it waits to receive the messages of Π^{M} generated bye the honest party P_i and replies with the messages computed by the malicious parties indexed by $[n] \setminus \{i\}$. In the reduction we have a challenger that, having black box access to $\mathcal{M}_0(\mathcal{A}, \cdot)$, either interacts with it using the messages of Π^{M} generated accordingly to the honest procedure or using the simulator $\mathcal{S}'$, which exists by the security definition. We note that in the case where the messages are generated accordingly to the honest procedure that the output of $\mathcal{M}_0(\mathcal{A}, \cdot)$ corresponds to the output of H_0, otherwise it corresponds to the output of H_1.

Besides showing that the hybrids H_0 and H_1 are indistinguishable, we also need to show that the outputs of the protocol in both of the hybrids is correct with respect to its inputs. This means that we need to show that $C(x_1, \dots, x_n) = \mathsf{Dec}(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \mathsf{Eval}(C, \mathsf{ct}_1^1, \dots, \mathsf{ct}_n^n))$ for all possible inputs $(x_i, \mathsf{ct}^i, K^i := (\mathsf{pk}_j^i, \mathsf{ct}_j^i)_{j\in[n]}, \mathsf{sk}_i, r_i^{\mathsf{Setup}}, r_i^{\mathsf{Enc}})_{i\in[n]}$ that pass all the tests described in Fig. 25. We prove this by contradiction. Now, we assume that $C(x_1, \dots, x_n)$ is unequal to the output of the MPC protocol $\mathsf{Dec}(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \mathsf{Eval}(C, \mathsf{ct}_1^1, \dots, \mathsf{ct}_n^n))$. In more detail, $C(x_1, \dots, x_n) \neq \mathsf{Dec}(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \mathsf{Eval}(C, \mathsf{ct}_1^1, \dots, \mathsf{ct}_n^n))$, where $\mathsf{ct}_i^i := \mathsf{Enc}(\mathsf{pk}_i^i, x_i; r_i^{\mathsf{Enc}})$ with $(\mathsf{pk}_i^i, \mathsf{sk}_i) := \mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$. This directly yields a contradiction to the perfect correctness of the multi-key FHE scheme (Definition 8) which states that $C(x_1, \dots, x_n) = \mathsf{Dec}(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \mathsf{Eval}(C, \mathsf{ct}_1^1, \dots, \mathsf{ct}_n^n))$, where $\mathsf{ct}_i^i := \mathsf{Enc}(\mathsf{pk}_i^i, x_i; r_i^{\mathsf{Enc}})$ for any x_i and any $r_i^{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$ with $(\mathsf{pk}_i, \mathsf{sk}_i) := \mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$ for any $r_i^{\mathsf{Setup}} \leftarrow \{0,1\}^\lambda$. □

Claim 6 (Transition from H_1 to H_2) *If* MFHE *is a semantic secure perfectly correct multi-key fully homomorphic encryption scheme, then the output distribution of the hybrids* H_1 *and* H_2 *are computationally indistinguishable.*

Proof. The difference between H_1 and H_2 is that the encryption of x_i sent in the second round is replaced by an encryption of 0. Assuming that the output distribution of H_1 and H_2 are distinguishable, we can construct an adversary $\mathcal{A}'$ that breaks the semantic security of MFHE. The adversary $\mathcal{A}'$ works as follows.

1. Receive pk_i from the challenger $\mathcal{C}$.
2. Send $(x_i, 0)$ to the challenger $\mathcal{C}$ and receive ct_i as a reply.
3. Act exactly as in H_1 (and H_2) with the following differences:
   (a) In round one output the keys pk_i and the ciphertext ct_i received from the challenger $\mathcal{C}$.
   (b) For every query $(x_j, \mathsf{ct}^j, K^j, \mathsf{sk}_j, r_j^{\mathsf{Setup}}, r_j^{\mathsf{Enc}})_{j\in I}$ asked by the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$, check that $(\mathsf{pk}_j^j, \mathsf{sk}_j^j) := \mathsf{Setup}(1^\lambda; r_j^{\mathsf{Setup}}), \mathsf{ct}_j^j := \mathsf{Enc}(\mathsf{pk}_j, x_j; r_j^{\mathsf{Enc}})$ for all $j \in I$ with $K^j = (\mathsf{pk}_l^j, \mathsf{ct}_l^j)_{l\in[n]}$ and check that $(\mathsf{pk}_l^i, \mathsf{ct}_l^i) = (\mathsf{pk}_l^j, \mathsf{ct}_l^j)$ for all $j \in I, l \in [n]$. Compute $\mathsf{ct} := \mathsf{Eval}(C, (\mathsf{pk}_1^i, \mathsf{ct}_1^i), \dots, (\mathsf{pk}_n^i, \mathsf{ct}_n^i)), y := \mathsf{Dec}(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \mathsf{ct})$, $y^j := \mathsf{Dec}(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \mathsf{ct}^j)$ and check that $y = y^j$ for all $j \in I$. If one of these checks fails, **Abort**. In the case that these tests pass, query the ideal functionality C using $\{x_j\}_{j\in I}$, receive $y := C(x_1, \dots, x_n)$ as an output and send y to $\Pi^{\mathsf{M}}.\mathcal{S}$.
   (c) Output what $\mathcal{A}$ outputs.

To ensure that the adversary $\mathcal{A}$ does not have a distinguishing advantage between the hybrids H_1 and H_2, we need to show that the output of the protocol in H_1 and H_2 is the same. This means we need to show that $C(x_1, \dots, x_n) \neq \mathsf{Dec}(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \mathsf{Eval}(C, (\mathsf{pk}_1^i, \mathsf{ct}_1^i), \dots, (\mathsf{pk}_n^i, \mathsf{ct}_n^i))$ for all possible inputs $(x_i, \mathsf{ct}^i, K^i := (\mathsf{pk}_j^i, \mathsf{ct}_j^i)_{j\in[n]}, \mathsf{sk}_i, r_i^{\mathsf{Setup}}, r_i^{\mathsf{Enc}})_{i\in[n]}$ to the protocol Π^{M} that pass all the checks. Now, we assume that this is not the case and that there exists an input $(x_i, \mathsf{ct}^i, K^i := (\mathsf{pk}^i_j, \mathsf{ct}^i_j)_{j\in[n]}, \mathsf{sk}_i, r_i^{\mathsf{Setup}}, r_i^{\mathsf{Enc}})_{i\in[n]}$ to the protocol Π^{M} that pass all the checks, such that $C(x_1, \ldots, x_n) \neq \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{Eval}(C, (\mathsf{pk}^i_1, \mathsf{ct}^i_1), \ldots, (\mathsf{pk}^i_n, \mathsf{ct}^i_n))$, but this would yield a contradiction to the perfect correctness of the multi-key FHE scheme (Definition 8) as described in the proof of Claim 5. This results in the fact that the output of the protocol in H_1 and H_2 is the same for every possible correct input. This also results in the correctness of the outputs in both hybrids as in Claim 6

The proof ends with the observation that the output of $\mathcal{A}''$ in the case where $\mathcal{C}$ has encrypted x_i corresponds to the output of H_1, and to the output of H_2 otherwise. □

$\underline{\Pi^{\mathsf{FHE}}}$

Initialization: Each $i \in [n]$ party P_i has input $x_i \in \{0,1\}^*$ as its secret input. We initialize τ_0 to 1^λ.

Round 1.
1. Compute $\mathsf{msg}_{1,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Round}_1(1^\lambda)$.
2. Sample $r_i^{\mathsf{Setup}} \leftarrow \{0,1\}^\lambda$ and compute $(\mathsf{pk}_i, \mathsf{sk}_i) := \mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$.
3. Sample $r_i^{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$ and compute $\mathsf{ct}_i := \mathsf{Enc}(\mathsf{pk}_i, x_i; r_i^{\mathsf{Enc}})$.
4. Send $(\mathsf{msg}_{1,i}, \mathsf{pk}_i, \mathsf{ct}_i)$.

Round 2.
1. Let τ_1 denote the transcript of the protocol Π^{M} up to round 1.
2. Compute $\mathsf{ct}^i := \mathsf{Eval}(C, (\mathsf{pk}_1, \mathsf{ct}_1), \ldots, (\mathsf{pk}_n, \mathsf{ct}_n))$.
3. Compute $\mathsf{msg}_{2,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Round}_2(y_i, \tau_1)$, where $y_i := (x_i, \mathsf{ct}^i, K^i, \mathsf{sk}_i, r_i^{\mathsf{Setup}}, r_i^{\mathsf{Enc}})$ and $K^i := (\mathsf{pk}_j, \mathsf{ct}_j)_{j\in[n]}$.
4. Send $\mathsf{msg}_{2,i}$.

For each round $k \in \{3, \ldots, \ell\}$.
1. Let τ_{k-1} denote the transcript of the protocol Π^{M} up to round $k-1$.
2. Compute the k-th round message $\mathsf{msg}_{k,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Round}_k(\tau_{k-1})$.
3. Send $\mathsf{msg}_{k,i}$.

Output Computation.
1. Let τ_ℓ denote the transcript of the protocol Π^{M} up to round ℓ.
2. Compute the output of Π^{M} as $y \leftarrow \Pi^{\mathsf{M}}.\mathsf{Out}(\tau_\ell)$.
3. Output y.

Fig. 24: Description of the protocol Π^{FHE} that securely realizes any functionality with knowledge of outputs.

Input: $(x_i, \mathsf{ct}^i, K^i, \mathsf{sk}_i, r_i^{\mathsf{Setup}}, r_i^{\mathsf{Enc}})_{i\in[n]}$.
- For all $i \in [n]$, parse K^i as $(\mathsf{pk}^i_j, \mathsf{ct}^i_j)_{j\in[n]}$.
- For all $i, j \in [n]$ check that $(\mathsf{pk}^i_i, \mathsf{ct}^i_i) = (\mathsf{pk}^j_i, \mathsf{ct}^j_i)$.
- For all $i \in [n]$ check that $(\mathsf{pk}_i, \mathsf{sk}_i) := \mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$ and $\mathsf{ct}^i_i := \mathsf{Enc}(\mathsf{pk}_i, x_i; r_i^{\mathsf{Enc}})$.
- Compute $y^i := \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{ct}^i)$ and check that $y^i = y^j$ for all $i, j \in [n]$.

If one of the above checks fails then output $\perp$, continue as follows otherwise.
Set y to y^i for a random $i \in [n]$.

Output: y to P_i.

Fig. 25: Circuit C_{Dec}

The following theorem follows immediately from Theorem 5 and the definition of a compact multi key FHE scheme.

Theorem 6. *Let* MFHE *be a compact multi key FHE scheme, then* Π^{FHE} *is a circuit independent secure MPC protocol that realizes any single-output functionality.*

Additionally, we can easily modify Π^{FHE} and obtain a protocol $\Pi^{\mathsf{FHE}'}$ which has communication complexity $O(L_{\mathsf{in}}) + \mathrm{poly}(\lambda, n, L_{\mathsf{out}})$. $\Pi^{\mathsf{FHE}'}$ works exactly as Π^{FHE} with the following differences. Every party P_i encrypts a short seed s_i of a PRG PRG using the FHE scheme, i.e. $\mathsf{Enc}(\mathsf{pk}_i, s_i; r_i^s)$, and sends it together with the value $w_i = \mathsf{PRG}(s_i) \oplus x_i$ to all the other parties P_j with $j \in [n] \setminus \{i\}$. The party P_i, upon receiving $(\mathsf{Enc}(\mathsf{pk}_j, s; r_j^s), w_j)$ from all the other parties P_j with $j \in [n] \setminus \{i\}$, computes $\mathsf{Enc}(\mathsf{pk}_j, \mathsf{PRG}(s_j))$, using homomorphic operations, $\mathsf{Enc}(\mathsf{pk}_j, w_j)$ by encrypting w_j using pk_j, and then homomorphically XORs the resulting ciphertexts to receive $\mathsf{Enc}(\mathsf{pk}_j, x_j)$. This ciphertext can now be used to run the evaluation algorithm and compute $\mathsf{Enc}(\{\mathsf{pk}_j\}, f(x_1, \ldots, x_n))$. The parties now check that the ciphertexts $\{w_j\}_{j \in [n]}$ are well formed by running the MPC protocol as described in Fig. 25.

Theorem 7. *Let* MFHE *be a compact multi key FHE scheme, then* $\Pi^{\mathsf{FHE}'}$ *is a secure MPC protocol with communication complexity* $O(L_{\mathsf{in}}) + \mathrm{poly}(\lambda, n, L_{\mathsf{out}})$ *that realizes any single-output functionality.*

Instantiations. To instantiate the underlying multi key FHE scheme, we can rely on the work of López-Alt et al. [LTV12]. In their work, they present several schemes. The first scheme is a multi key fully homomorphic encryption scheme for a constant number of party that allows for the evaluation of any circuit based on a perfectly correct FHE scheme. As an instantiation for the perfectly correct FHE scheme we could for example use [BGV12] which can be either based on LWE or ring-LWE. Additionally, López-Alt et al. present a multi key FHE scheme for any number of parties based on the Decisional Small Polynomial Ratio (DSPR) and the ring-LWE assumption.

By relying on the compiler that amplifies security from privacy with knowledge of outputs to full security, based on signatures (Section 6), the compiler that realizes any functionality from single-output functionalities, based on symmetric encryption (Section 7), and the facts that a signature scheme can be obtained from one way functions [Rom90] and that a one-time symmetric encryption scheme can be instantiate information-theoretically, we obtain the following corollary.

Corollary 2. *If the LWE and DSPR assumptions hold and any of the DDH, QR, N^{th} Residuosity holds, or there exists a malicious-secure OT, then there exists a round optimal (4-round)* circuit-independent *MPC protocol that realizes any functionality.*

6 From Privacy with Knowledge of Outputs to Standard Security

We recall that a protocol that realizes a functionality f without knowledge of outputs allows the adversary to see the output of the computation y, and then lets the adversary decide what the output of the honest parties should be. We can rely on the results of [IKP10] and [PC12] where the authors present a compiler that turns a protocol Π^{PKO} that realizes any *single-output* function under security with knowledge of outputs, into a protocol Π^{Corr} that securely realizes any single-output function in the standard simulation based sense. In this section, we recap the compiler of [IKP10] and [PC12] as well as their security proof. Since Ishai et al. already have shown that the compiler preserves the round complexity, we only need to argue that it also preserves the commmunication complexity to the underlying protocol.

Before we formally define the compiler, we present an informal description and a proof intuition of the protocol.

6.1 Informal Description

To turn a protocol with knowledge of outputs Π^{PKO} that realizes the circuit C into a protocol Π^{Corr} that achieves standard simulation based security for the same circuit, we make use of a signature scheme DS. In more detail, every party P_i will sample a verification and signing key $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$ and use the key sk_i together with a random value r_i and the inputs x_i as the input to the MPC protocol Π^{PKO}. In addition, each party sends its verification key vk_i. The circuit that the MPC protocol Π^{PKO} evaluates consists of two steps. In the first step, it computes the circuit C on the inputs $(x_1, \ldots, x_n)$ and generates $y := C(x_1, \ldots, x_n)$. In the last step, the output y is signed under the different signing keys, i.e. the signatures $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, y; r_i)$ are generated for all $i \in [n]$. The output of the MPC protocol Π^{PKO} then corresponds to the output y and all the signatures $\{\sigma_i\}_{i\in[n]}$ generated under the signing keys $\{\mathsf{sk}_i\}_{i\in[n]}$ of all the parties P_i. Finally, every party P_i uses the received verification keys $\{\mathsf{vk}_i\}_{i\in[n]}$ to verify all the signatures, i.e. it computes $b_i \leftarrow \mathsf{Verify}(\mathsf{vk}_i, y, \sigma_i)$ for all $i \in [n]$. If one of the values b_i is equal to 0, then the honest parties would abort. The unforgeability of the digital signature scheme DS makes sure that no party is able to create signatures for the verification keys of an honest party. Intuitively, this means that an adversary that receives the output cannot tamper with it unless it can break the security of the signature scheme.

6.2 Formal Description

Now, we describe the protocol Π^{Corr} formally. We start by describing the building blocks used for the construction of Π^{Corr}.

Bulding Blocks. Let C be the single-input function that we want to securely evaluate. The tools that we use to construct our protocol, which we denote with Π^{Corr}, are the following.

- A signature scheme $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$
- A protocol Π^{PKO} that realizes the function C_σ with knowledge of outputs described in Fig. 27.

We refer to Fig. 26 for the formal description of Π^{Corr}.

$\underline{\Pi^{\mathsf{Corr}}}$

Each party P_i, on input x_i does the following steps:

1. Compute $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$.
2. Run the protocol Π^{PKO} using as input $(x_i, \mathsf{sk}_i, r_i)$ for a random r_i. In addition, send vk_i in the first round.
3. Upon receiving the output $(y, \sigma_k)_{k\in[n]}$ of Π^{PKO} do the following
   (a) Let $(\mathsf{vk}_j)_{j\in[n]\setminus\{i\}}$ be the public keys received in the first round from the parties $(P_j)_{j\in[n]\setminus\{i\}}$ respectively.
   (b) If for each $j \in [n]$ $\mathsf{Verify}(\mathsf{vk}_j, y, \sigma_j) = 1$ then output c, otherwise output $\perp$.

Fig. 26: Our compiler: from MPC with knowledge of outputs to MPC with correctness.

Input: $(x_i, \mathsf{sk}_i, r_i)_{i\in[n]}$
Compute $y := C(x_1, \ldots, x_n)$ and
$\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, y; r_i)$ for all $i \in [n]$
Output: $(y, (\sigma_i)_{i\in[n]})$.

Fig. 27: Circuit C_σ

Theorem 8. *Let C be an n-party single-output randomized functionality, if DS is a signature scheme and Π^{PKO} realizes the function C_σ with knowledge of outputs then Π^{Corr} securely realizes C.*

Proof. To prove our lemma we need to show that for every PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $I \subset [n]$ the following quantity is negligible:

$$|\Pr[\mathrm{Real}_{\Pi^{\mathsf{Corr}},\mathcal{A}(z),I}(k,\boldsymbol{x})=1]-\Pr[\mathrm{Ideal}_{C,\mathcal{S}(z),I}(k,\boldsymbol{x})=1]|\ ,$$

where $\boldsymbol{x}=\{x_i\}_{i\in[n]}\in\{0,1\}^*$ and $z\in\{0,1\}^*$ and I denotes the set that contains the indices of all the corrupted parties, i.e. $|I|=n-1$. Also in this case, for simplicity, we assume that all but one of the parties is corrupted. Before describing how $\mathcal{S}$ works, we define an algorithm $\mathcal{M}$. The augmented machine internally runs the adversary $\mathcal{A}$ (we refer to this interaction as the right session), and acts as a proxy between $\mathcal{A}$ and its external interface with respect to the messages of Π^{PKO} (that we call left session). To describe our simulator we need to described the augmented machine $\mathcal{M}$ and its interaction with $\mathcal{S}$ (i.e., how the messages of Π^{PKO} are computed).

The reason why we describe our simulator using the augmented machine $\mathcal{M}$ is to deal with the unknown actions that the simulator of Π^{PKO} might execute (e.g., rewinds). More precisely, the augmented machine $\mathcal{M}$ acts as an adversary for the protocol Π^{PKO}.

By assumption, we know that for every PPT adversary $\mathcal{A}'$ there exists a PPT adversary $\mathcal{S}'$ such that for any $I \subset [n]$ the following quantity is negligible:

$$|\Pr[\mathrm{Real}_{\Pi^{\mathsf{PKO}},\mathcal{A}(z),I}(\lambda,\boldsymbol{x})=1]-\Pr[\mathrm{Ideal}^{\mathsf{PKO}}_{C_\sigma,\mathcal{S}'(z),I}(\lambda,\boldsymbol{x})=1]|$$

where $\boldsymbol{x}=\{x_i\}_{i\in[n]}\in\{0,1\}^*$ and $z\in\{0,1\}^*$.

Therefore, we can run the simulator $\mathcal{S}'$ for the adversary $\mathcal{M}$. For the formal description of $\mathcal{M}$ we refer to Fig. 28 and for the formal description of $\mathcal{S}$ we refer to Fig. 29.

```
$\mathcal{M}(r_{\mathcal{A}})$ :
For round 1.
    1. Compute $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$.
    2. Upon receiving the message $\mathsf{msg}_{1,i}$ from the left session send $(\mathsf{msg}_{i,1}, \mathsf{vk}_i)$ to $\mathcal{A}$.
    3. Receive the messages $(\mathsf{msg}_{1,j}, \mathsf{vk}_j)_{j\in I}$ and forward $\{\mathsf{msg}_{1,j}\}_{j\in I}$ in the left session.
For each Round $k \in \{2,\dots,\ell\}$.
    1. Upon receiving the message $\mathsf{msg}_{k,i}$ from the left session forward it to $\mathcal{A}$.
    2. Receive the messages $\{\mathsf{msg}_{k,j}\}_{j\in I}$ and forward them in the left session.
```

Fig. 28: The augmented machine $\mathcal{M}$ which emulates the adversary for Π^{PKO}.

```
$\mathcal{S}$
 – Sample a sufficiently long random $R$ and run the simulator $\Pi^{\mathsf{PKO}}.\mathcal{S}$ for the adversary $\mathcal{M}$.
 – For every query $(x_j, \mathsf{sk}_j, r_j)_{j\in I}$ issued by $\Pi^{\mathsf{M}}.\mathsf{PKO}$ do the following:
    1. Query the ideal functionality using $\{x_j\}_{j\in I}$ and receive $(y_1,\dots,y_n) = C(x_1,\dots,x_n)$ as an output.
    2. Sample a random value $r_i \leftarrow \{0,1\}^\lambda$.
    3. Compute $\sigma_k \leftarrow \mathsf{Sign}(\mathsf{sk}_k, x_k; r_k)$ for all $k \in [n]$ and output $\{y_i, \sigma_k\}_{k\in[n]}$.
 – When $\Pi^{\mathsf{one}}.\mathcal{S}$ stops, output what $\Pi^{\mathsf{one}}.\mathcal{S}$ outputs.
```

Fig. 29: The simulator $\mathcal{S}$ for our protocol Π^{Corr}.

Hybrid H_0: Hybrid H_0 is identical to the real world experiment

Hybrid H_1: . By assumption, we know that for every PPT adversary $\mathcal{A}'$ there exists a PPT adversary $\mathcal{S}'$ such that for any $I \subset [n]$ the following quantity is negligible:

$$|\Pr[\mathrm{Real}_{\Pi^{\mathsf{PKO}},\mathcal{A}'(z),I}(\lambda, \boldsymbol{x}) = 1] - \Pr[\mathrm{Ideal}^{\mathsf{PKO}}_{C_\sigma,\mathcal{S}'(z),I}(\lambda, \boldsymbol{x}) = 1]|$$

where $\boldsymbol{x} = \{x_i\}_{i\in[n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$. In hybrid H_1 the messages of Π^{PKO} are simulated using $\mathcal{S}'$. In more details, H_1 runs $\mathcal{S}'$ and the adversary $\mathcal{A}$, and acts exactly as in H_0 with the following differences.

1. All the messages of Π^{PKO} are replaced with the messages of $\mathcal{S}'$.
2. Upon receiving the query $(x_j, \mathsf{sk}_j, r_j)_{j\in I}$ from $\mathcal{S}'$ (who wants to query the ideal world functionality to evaluate C_σ), compute $c := C(x_1, \ldots, x_n)$.
3. For each $k \in [n]$ compute $\sigma_k \leftarrow \mathsf{Sign}(\mathsf{sk}_k, c; r_k)$ and send $(c, (\sigma_k)_{k\in[n]})$ to $\mathcal{S}'$.
4. Let $\{\mathsf{vk}_k\}_{k\in n}$ be the verification keys sent in the first round respectively by $P_1, \ldots, P_n$ and $(c', (\sigma'_k)_{k\in[n]})$ be the output computed by $\mathcal{S}'$ for the honest parties. If for all $k \in [n]$ $\mathsf{Verify}(\mathsf{vk}_k, c', \sigma'_k) = 1$ then output c', otherwise output $\perp$.

We show that the output distributions of the two hybrids is indistinguishable, and then prove that the output received by the honest parties in both hybrids is correct.

Lemma 5 (Transition from H_0 to H_1). *Let Π^{PKO} be a maliciously secure MPC protocol, then the output distributions of the hybrids H_0 and H_1 are computationally indistinguishable.*

Proof. By contradiction, we assume that the output distributions of H_0 and H_1 are distinguishable by a non-negligible quantity p. If this is the case, then we can construct an adversary $\mathcal{A}'$ that breaks the security of Π^{PKO}. The adversary $\mathcal{A}'$ works exactly as in H_0 (and H_1) with the difference that it acts as a proxy with respect to all the messages of Π^{PKO} between $\mathcal{A}$ and an external challenger. The external challenger tosses a coin b', and if $b' = 0$ then the messages of Π^{PKO} are computed accordingly to the honest procedure, otherwise those messages are computed accordingly to $\mathcal{S}'$. The output of $\mathcal{A}'$ corresponds to the output of $\mathcal{A}$. We note that in the case where $b' = 0$ the output of $\mathcal{A}'$ corresponds to the output of $\mathcal{A}$ in H_0, and to the output of $\mathcal{A}$ in H_1 otherwise. □

We define the event $\mathsf{WrongOutput}_{\mathsf{H}_b}$ as the event in which the output computed by the honest party is incorrect (i.e., P_i accepts $c \neq c' := C(x_1, \ldots x_n)$ as a valid output) in the hybrid $b \in \{0,1\}$.

We now prove the following lemma.

Lemma 6. $|\Pr[\mathsf{WrongOutput}_{\mathsf{H}_0}] - \Pr[\mathsf{WrongOutput}_{\mathsf{H}_1}]| \leq \mathrm{negl}(\lambda)$.

The proof of this lemma follows immediately from Lemma 5

What it remains to is the following lemma.

Lemma 7. $\Pr[\mathsf{WrongOutput}_{\mathsf{H}_1}] \leq \mathrm{negl}(\lambda)$.

Proof. Assume by contradiction that $\Pr[\mathsf{WrongOutput}_{\mathsf{H}_1}]$ is equal to a non-negligible quantity p, then we can construct an adversary $\mathcal{A}^{\mathsf{DS}}$ that breaks the security of the digital signature scheme DS. The adversary $\mathcal{A}^{\mathsf{DS}}$ receives the a verification key pk from the challenger of the unforgeability security game, and on input x_i acts as follows.

1. Run accordingly to H_1 and upon receiving the query $(x_j, \mathsf{sk}_j, r_j)_{j\in I}$ from $\mathcal{S}'$ (who wants to query the ideal world functionality to evaluate C_σ), compute $c := C(x_1, \ldots, x_n)$.
2. For each $j \in [n] \setminus \{i\}$ compute $\sigma_j \leftarrow \mathsf{Sign}(\mathsf{sk}_j, c)$, query the signing oracle $\mathsf{Sign}(\mathsf{sk}_i, \cdot)$ on input c thus obtaining the signature σ_i.

3. Send $(c, (\sigma_j)_{j\in[n]})$ to $\mathcal{S}'$.
4. Let $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ be the verification keys sent in the first round respectively by $P_1, \ldots, P_n$ and c', σ'_i be the output computed by $\mathcal{S}'$ for the honest party P_i. If $\mathsf{Verify}(\mathsf{pk}_i, c', \sigma'_i) = 1$ with $c \neq c'$ then output (σ'_i, c') as the forgery and stop, otherwise output $\perp$ and stop.

Having assumed by contradiction that $\Pr[\mathsf{WrongOutput}_{\mathsf{H}_1}]$ is equal to a non-negligible quantity p, then $\mathcal{A}^{\mathsf{DS}}$ is able to forge the signature scheme DS with non-negligible probability p. □

The proof ends with the observation that the output distribution of H_1 is identical to the one of $\mathcal{S}$ (which is described in Fig. 29) and that $\mathcal{S}$ never uses the input of the honest party P_i. □

To analyze the communication complexity of the resulting protocol Π^{PKO}, we define the input and output size of C with L_{in} and L_{out} and the communication complexity of Π^{PKO} with CP when the circuit being evaluated is C_σ. We can immediately conclude that the input size of C_σ is $L'_{\mathsf{in}} := \text{poly}(\lambda, n, L_{\mathsf{in}})$ and the outputs size is $L'_{\mathsf{out}} := \text{poly}(\lambda, n, L_{\mathsf{out}})$. Now we can state our theorem.

Theorem 9. *If Π^{PKO} has communication complexity CP when evaluating the circuit C_σ, then Π^{Corr} has communication complexity $\text{poly}(\lambda, n, \mathsf{CP}, L'_{\mathsf{in}}, L'_{\mathsf{out}}) = \text{poly}(\lambda, n, \mathsf{CP}, L_{\mathsf{in}}, L_{\mathsf{out}})$ when evaluating C.*

7 Individual Outputs for each party

Besides achieving security with correctness from a protocol with knowledge of output security, we need to show how to turn a protocol for single output functionalities Π^{one} (i.e., a protocol that provides the same output to all the parties), into a protocol that realizes any functionality Π^{many}. Here, we can rely on the compiler presented in [LP09, Section 2]. As in the previous section, we recap here the compiler of [LP09] and their security proof. Additionally, we show that also this compiler preserves the round and the communication complexity of the input protocol.

Before we formally define the compiler, we present an informal description and a proof intuition of the protocol.

7.1 Informal Description

Let $P_1, \ldots, P_n$ be the set of parties and $x_1, \ldots, x_n$ their respective inputs. Let $(y_1, \ldots, y_n) := C(x_1, \ldots, x_n)$ be the function that these parties want to compute in such a way that each party P_i learns only y_i and nothing beyond that. Our compiler makes use of Π^{one} and of a symmetric encryption scheme SE.

P_i samples a random value r_i, generates a symmetric encryption key $\mathsf{k}_i \leftarrow \mathsf{Setup}(1^\lambda)$ and uses (k_i, r_i, x_i) as the input to the MPC protocol Π^{one}. Π^{one} evaluates the circuit C_{many} that 1) computes the circuit C on the inputs $(x_1, \ldots, x_n)$ and generates $(y_1, \ldots, y_n) := C(x_1, \ldots, x_n)$. In the last step, the outputs $(y_1, \ldots, y_n)$ are encrypted using the different keys, i.e., the ciphertexts $c_i \leftarrow \mathsf{Enc}(\mathsf{k}_i, y_i; r_i)$ are generated for all $i \in [n]$. The output of the MPC protocol Π^{one} then consists of all the ciphertexts $\{c_i\}_{i\in[n]}$ (we recall that the same set of cipthertexts is sent to all parties). Finally, every party P_i uses its secret key k_i to decrypt the ciphertext c_i thus obtaining y_i. The security of the symmetric encryption scheme ensures that no party learns anything about the outputs of the other parties. We also note that if the communication complexity of Π^{one} is CT then the communication complexity of Π^{many} is $\text{poly}(\lambda, n, \mathsf{CT})$.

This concludes the description of the protocol Π^{many}.

7.2 Formal Description

Now, we describe the protocol Π^{many} formally. We start by describing the building blocks used for the construction of Π^{many}.

Building Blocks. Our Construction makes use of the following cryptographic tools:

- A secret key encryption scheme $\mathsf{SE} := (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$
- A protocol Π^{one} that realizes the circuit C_{many} described in Fig. 31.

A formal description of the protocol Π^{many} is presented below.

$\underline{\Pi^{\mathsf{many}}}$

Each party P_i, on input x_i executes the following steps:

1. Compute $\mathsf{k}_i \leftarrow \mathsf{Setup}(1^\lambda)$.
2. Run the protocol Π^{one} using input (x_i, k_i, r_i) for a random value r_i.
3. Upon receiving the output $\{c_j\}_{j\in[n]}$ of Π^{one}, compute $y_i = \mathsf{Dec}(\mathsf{k}_i, c_i)$ and output y_i.

Fig. 30: Description of the protocol Π^{many}.

Input: $(x_i, \mathsf{k}_i.r_i)_{i\in[n]}$

Compute $(y_1, \ldots, y_n) := C(x_1, \ldots, x_n)$ and

$c_i \leftarrow \mathsf{Enc}(\mathsf{k}_i, y_i; r_i)$ for all $i \in [n]$

Output: $\{c_i\}_{i\in[n]}$.

Fig. 31: Circuit C_{many}

Theorem 10. *Let C be an n party many-output randomized functionality, if $\mathsf{SE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ is an* IND-CPA *secure secret key encryption scheme and Π^{one} realizes the function C_{many}, which is a single-output functionality, then Π^{many} securely realizes C.*

Proof. To prove our lemma we need to show that for every PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $I \subset [n]$ the following quantity is negligible:

$$|\Pr[\mathrm{Real}_{\Pi^{\mathsf{many}},\mathcal{A}(z),I}(k, \boldsymbol{x}) = 1] - \Pr[\mathrm{Ideal}_{C,\mathcal{S}(z),I}(k, \boldsymbol{x}) = 1]| \ ,$$

where $\boldsymbol{x} = \{x_i\}_{i\in[n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$.

Before describing how $\mathcal{S}$ works, we define an algorithm $\mathcal{M}$. The augmented machine internally runs the adversary $\mathcal{A}$ (we refer to this interaction the right session), and acts as a proxy between $\mathcal{A}$ and its external interface with respect to the messages of Π^{one} (to which we refer to in the left session). To describe our simulator we need to described the augmented machine $\mathcal{M}$ and its interaction with $\mathcal{S}$ (i.e., how the messages of Π^{one} are computed). More precisely, the augmented machine $\mathcal{M}$ acts as an adversary for the protocol Π^{one}. Hence, we can consider the simulator $\Pi^{\mathsf{one}}.\mathcal{S}$ for Π^{one} for the adversary $\mathcal{M}$. Our simulator $\mathcal{S}$ will then simply run $\Pi^{\mathsf{one}}.\mathcal{S}$. For the formal description of $\mathcal{M}$ we refer to Fig. 32 and for the formal description of $\mathcal{S}$ we refer to Fig. 33.

```
$\underline{\mathcal{M}(r_{\mathcal{A}}):}$
For each Round $k \in \{1, \ldots, \ell\}$.
    1. Upon receiving the message $\mathsf{msg}_{k,i}$ from the left session forward it to $\mathcal{A}$.
    2. Receive the messages $\{\mathsf{msg}_{k,j}\}_{j \in I}$ and forward them in the left session.
```

Fig. 32: The augmented machine $\mathcal{M}$ which emulates the adversary for Π^{one}.

```
$\underline{\mathcal{S}}$
– Sample a sufficiently long random $R$ and run the simulator $\Pi^{\mathsf{one}}.\mathcal{S}$ for the adversary $\mathcal{M}$.
– For every query $(x_j, \mathsf{k}_j, r_j)_{j \in I}$ issued by $\Pi^{\mathsf{M}}.\mathsf{one}$ do the following:
    1. Compute $\mathsf{k}_i \leftarrow \mathsf{Setup}(1^\lambda)$ and sample a random value $r_i \leftarrow \{0,1\}^\lambda$.
    2. Query the ideal functionality using $\{x_j\}_{j \in I}$ and receive $(y_1, \ldots, y_n) = C(x_1, \ldots, x_n)$ as an output.
    3. Compute $c_k \leftarrow \mathsf{Enc}(\mathsf{k}_k, x_k; r_k)$ for all $k \in [n]$ and output $\{c_k\}_{k \in [n]}$.
– When $\Pi^{\mathsf{one}}.\mathcal{S}$ stops, output what $\Pi^{\mathsf{one}}.\mathcal{S}$ outputs.
```

Fig. 33: The simulator $\mathcal{S}$ for our protocol Π^{many}.

We propose the description of a simulator $\mathcal{S}$ (Fig. 33), and consider the following sequence of hybrids to show that the real world is indistinguishable from the ideal world.

Hybrid H_0: Hybrid H_0 is identical to the real world experiment.

Hybrid H_1: In hybrid H_1, the inner MPC protocol Π^{M} is simulated instead of honestly generated. The transition between hybrid H_0 and H_1 is justified by the malicious security of the MPC protocol Π^{M} and formally proven in Lemma 8.

Hybrid H_2 This hybrid is identical to the ideal world. In this hybrid, the ciphertext that is an encryption of the output y_i for party P_i is exchanged with an encryption of a random value. The transition between hybrid H_1 and H_2 is justified by the IND-CPA security of the single key encryption scheme SE and formally proven in Lemma 9.

Putting everything together, we obtain the theorem. □

Lemma 8 (Transition from H_0 to H_1). *Let Π^{one} be a maliciously secure MPC protocol, then the hybrids H_0 and H_1 are computationally indistinguishable.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids with non-negligible probability. We use $\mathcal{A}$ to construct an augmented machine $\mathcal{M}$ that breaks the security of the underlying MPC protocol Π^{M}.

We describe the interactive machine $\mathcal{M}$ that interacts with the MPC protocol Π^{M} as all the corrupted parties in the left session and with an adversary $\mathcal{A}$ using randomness $r_{\mathcal{A}}$ in the right session. Afterwards we describe how to reply to ideal functionality queries asked by the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$. We stress that whenever the augmented machine is rewound by the inner MPC protocol it rewinds the adversary $\mathcal{A}$ it is interacting with accordingly and continues the execution with $\mathcal{A}$ using the randomness $r_{\mathcal{A}}$ afterwards.

The augmented machine $\mathcal{M}$ acts as an intermediator between the left and the right session. It forwards the messages it receives in the left session, which represent the messages of the party P_i, to the adversary $\mathcal{A}$ in the right session and it also forwards the messages that it receives in the right session, representing the messages of the corrupted parties P_j, to the left session.

To finish the description of the reduction, and to achieve consistency between the output of the inner MPC protocol Π^{one} and the interactions of the parties, we need to adjust the answers given to the ideal functionality queries by the simulator $\Pi^{\mathsf{one}}.\mathcal{S}$.

For every query $(x_j, \mathsf{k}_j, r_j)_{j \in I}$ generate a secret key $\mathsf{k}_i \leftarrow \mathsf{Setup}(1^\lambda)$ and sample a random value $r_i \leftarrow \{0,1\}^\lambda$ and compute $(y_1, \ldots, y_n) = C(x_1, \ldots, x_n)$. In the next step, the outputs are encrypted under the different secret keys, i.e. compute $c_k \leftarrow \mathsf{Enc}(\mathsf{k}_k, y_k; r_k)$ for all $k \in [n]$. Finally, send $\{c_k\}_{k \in [n]}$ as a reply to the simulator $\Pi^{\mathsf{one}}.\mathcal{S}$. This concludes the description of the simulator.

Since we assume that the adversary $\mathcal{A}$ is able to distinguish between the two hybrids H_0 and H_1 with non-negligible probability, it would be able to detect if the messages $\mathsf{msg}_{1,i}, \ldots, \mathsf{msg}_{4,i}$ of the underlying MPC are simulated or honestly generated. This would also allow the machine $\mathcal{M}$ to distinguish if the inner MPC Π^{M} has been simulated or honestly generated which contradicts the malicious security of the inner MPC Π^{M}. Therefore, the claim follows. □

Lemma 9 (Transition from H_1 to H_2). *Let* $\mathsf{SE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *be an* IND-CPA *secure secret key encryption scheme, then the hybrids* H_1 *and* H_2 *are computationally indistinguishable.*

Proof. In the transition from hybrid H_1 to hybrid H_2 the ciphertext c_i that is an encryption the output y_i for the party P_i is changed to an encryption of a random value r.

To describe the transition from H_1 to H_2, we need to define the answers to the simulator Π^{M} when it queries its ideal functionality. We do not need to change anything in the description of the augmented machine $\mathcal{M}$ since it behaves in both hybrids in the same way, with the only difference that the augmented machine does not need to send any input (x_i, k_i) since the protocol execution is simulated. We prove the indistinguishability of H_1 and H_2 with a reduction to the IND-CPA security of the symmetric encryption scheme SE. In more detail, we suppose that there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids with non-negligible probability and we use this adversary $\mathcal{A}$ together with the augmented machine $\mathcal{M}$ to break the IND-CPA security of the underlying secret key encryption scheme SE in a game with the challenger $\mathcal{C}$.

To finish the description of the reduction, and to achieve consistency between the output of the inner MPC protocol and the interactions of the parties, we need to adjust the answers given to the ideal functionality queries by the simulator $\Pi^{\mathsf{M}}.\mathcal{S}$.

For any ideal functionality query $(x_j, \mathsf{k}_j)_{j \in I}$ asked by the simulator Π^{one}, query the ideal functionality to receive $(y_1, \ldots, y_n) = C(x_1, \ldots, x_n)$ and encrypt the outputs for the corrupted parties P_j under the different secret keys, i.e. compute $c_j \leftarrow \mathsf{Enc}(\mathsf{k}_j, y_j)$ for all $j \in I$. For the output y_i belonging to the honest party P_i send the tuple (y_i, r) with a random value r to the underlying challenger $\mathcal{C}$. The challenger replies with the ciphertext c_i. Finally, send $\{c_k\}_{k \in [n]}$ as a reply to the simulator $\Pi^{\mathsf{one}}.\mathcal{S}$. This concludes the description of the reduction.

In the case that the challenger replies with an encryption of y_i, the experiment between $\mathcal{M}$ and $\mathcal{A}$ corresponds exactly to H_1 and when the challenger replies to $\mathcal{M}$ with an encryption of the random value r, the experiment corresponds exactly to H_3. Since we assume that the adversary $\mathcal{A}$ is able to distinguish between the two hybrids H_1 and H_2 with non-negligible probability, it would be able to detect if the encryption c_i output in the second round corresponds to the value y_i or r. This would also allow the machine $\mathcal{M}$ to distinguish between the values its underlying challenger has encrypted which contradicts the IND-CPA of the secret encryption scheme SE. This leads to a contradiction and therefore the claim follows. □

To analyze the communication complexity of the resulting protocol Π^{PKO}, we define the input and output size of C with L_{in} and L_{out} and the communication complexity of Π^{one} with CP' when the circuit being evaluated is C_{many}. We can immediately conclude that the input size of C_{many} is $L'_{\mathsf{in}} := \mathrm{poly}(\lambda, n, L_{\mathsf{in}})$ and the outputs size is $L'_{\mathsf{out}} := \mathrm{poly}(\lambda, n, L_{\mathsf{out}})$.

We state the theorem regarding the communication complexity of Π^{many} formally.

Theorem 11. *If Π^{one} has communication complexity CP' when evaluating the circuit C_{many}, then Π^{many} has communication complexity* $\mathrm{poly}(\lambda, n, \mathsf{CP}', L'_{\mathsf{in}}, L'_{\mathsf{out}}) = \mathrm{poly}(\lambda, n, \mathsf{CP}', L_{\mathsf{in}}, L_{\mathsf{out}})$ *when evaluating C.*

Acknowledgments. The first author is supported in party by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement 780477 (PRIVILEDGE). The second author is supported in part by DARPA under Cooperative Agreement No: HR0011-20-2-0025, NSF Grant CNS-2001096, US-Israel BSF grant 2015782, Google Faculty Award, JP Morgan Faculty Award, IBM Faculty Research Award, Xerox Faculty Research Award, OKAWA Foundation Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. The third author is supported in part by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement 780108 (FENTEC). The fourth author is supported in part by Sunday Group, Inc.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, the Department of Defense, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright annotation therein.

References

ABJ+19. P. Ananth, S. Badrinarayanan, A. Jain, N. Manohar, and A. Sahai. From FE combiners to secure MPC and back. In *TCC 2019, Part I*, *LNCS* 11891, pages 199–228. Springer, Heidelberg, December 2019. (Pages 1, 2, 4, 5, 11, 12, 13, and 34.)

AJJM20. P. Ananth, A. Jain, Z. Jin, and G. Malavolta. Multikey fhe in the plain model. Cryptology ePrint Archive, Report 2020/180, 2020. https://eprint.iacr.org/2020/180. (Pages 3 and 5.)

AJW11. G. Asharov, A. Jain, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. Cryptology ePrint Archive, Report 2011/613, 2011. http://eprint.iacr.org/2011/613. (Page 7.)

BGG+14. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT 2014*, *LNCS* 8441, pages 533–556. Springer, Heidelberg, May 2014. (Page 34.)

BGJ+18. S. Badrinarayanan, V. Goyal, A. Jain, Y. T. Kalai, D. Khurana, and A. Sahai. Promise zero knowledge and its applications to round optimal MPC. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 459–487. Springer, Heidelberg, August 2018. (Pages 1, 3, 5, 7, and 34.)

BGV12. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325. ACM, January 2012. (Pages 2, 34, and 41.)

BL18. F. Benhamouda and H. Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *EUROCRYPT 2018, Part II*, *LNCS* 10821, pages 500–532. Springer, Heidelberg, April / May 2018. (Pages 1, 5, and 7.)

BMR90. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990. (Page 1.)

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC 2011*, *LNCS* 6597, pages 253–273. Springer, Heidelberg, March 2011. (Pages 2 and 10.)

Can03. R. Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. http://eprint.iacr.org/2003/239. (Page 16.)

CCG+19. A. R. Choudhuri, M. Ciampi, V. Goyal, A. Jain, and R. Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. Cryptology ePrint Archive, Report 2019/216, 2019. https://eprint.iacr.org/2019/216. (Pages 1, 3, 5, 7, and 34.)

DHRW16. Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 93–122. Springer, Heidelberg, August 2016. (Page 5.)

GB96. S. Goldwasser and M. Bellare. Lecture notes on cryptography. *Summer course "Cryptography and computer security" at MIT*, 1999:1999, 1996. (Page 14.)

GKP^{+}13. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *45th ACM STOC*, pages 555–564. ACM Press, June 2013. (Page 34.)

GMPP16. S. Garg, P. Mukherjee, O. Pandey, and A. Polychroniadou. The exact round complexity of secure computation. In *EUROCRYPT 2016, Part II*, *LNCS* 9666, pages 448–476. Springer, Heidelberg, May 2016. (Page 1.)

GMW87. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987. (Page 1.)

Gol04. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004. (Pages 15, 17, and 18.)

Goy11. V. Goyal. Constant round non-malleable protocols using one way functions. In *43rd ACM STOC*, pages 695–704. ACM Press, June 2011. (Page 1.)

GS17. S. Garg and A. Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In *58th FOCS*, pages 588–599. IEEE Computer Society Press, October 2017. (Page 5.)

GS18. S. Garg and A. Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *EUROCRYPT 2018, Part II*, *LNCS* 10821, pages 468–499. Springer, Heidelberg, April / May 2018. (Pages 1 and 5.)

GSW13. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO 2013, Part I*, *LNCS* 8042, pages 75–92. Springer, Heidelberg, August 2013. (Page 34.)

GVW15. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In *CRYPTO 2015, Part II*, *LNCS* 9216, pages 503–523. Springer, Heidelberg, August 2015. (Page 34.)

HHPV18. S. Halevi, C. Hazay, A. Polychroniadou, and M. Venkitasubramaniam. Round-optimal secure multi-party computation. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 488–520. Springer, Heidelberg, August 2018. (Pages 1 and 5.)

IKP10. Y. Ishai, E. Kushilevitz, and A. Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO 2010*, *LNCS* 6223, pages 577–594. Springer, Heidelberg, August 2010. (Pages 6, 7, 19, and 41.)

IPS08. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO 2008*, *LNCS* 5157, pages 572–591. Springer, Heidelberg, August 2008. (Page 1.)

Kil88. J. Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988. (Page 1.)

KO04. J. Katz and R. Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO 2004*, *LNCS* 3152, pages 335–354. Springer, Heidelberg, August 2004. (Page 1.)

KOS03. J. Katz, R. Ostrovsky, and A. Smith. Round efficiency of multi-party computation with a dishonest majority. In *EUROCRYPT 2003*, *LNCS* 2656, pages 578–595. Springer, Heidelberg, May 2003. (Page 1.)

Lin10. Y. Lindell. Foundations of cryptography 89-856. http://u.cs.biu.ac.il/~lindell/89-856/complete-89-856.pdf, 2010. (Page 16.)

LP09. Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009. (Pages 7 and 45.)

LTV12. A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012. (Pages 2, 4, 13, 14, and 41.)

MPP20. A. Morgan, R. Pass, and A. Polychroniadou. Succinct non-interactive secure computation. In *EUROCRYPT 2020, Part II*, *LNCS* 12106, pages 216–245. Springer, Heidelberg, May 2020. (Page 1.)

O'N10. A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/2010/556. (Pages 2 and 10.)

Pas04. R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *36th ACM STOC*, pages 232–241. ACM Press, June 2004. (Page 1.)

PC12. A. Paskin-Cherniavsky. *Secure computation with minimal interaction*. PhD thesis, Computer Science Department, Technion, 2012. (Pages 6, 19, and 41.)

PW10. R. Pass and H. Wee. Constant-round non-malleable commitments from sub-exponential one-way functions. In *EUROCRYPT 2010*, *LNCS* 6110, pages 638–655. Springer, Heidelberg, May / June 2010. (Page 1.)

QWW18. W. Quach, H. Wee, and D. Wichs. Laconic function evaluation and applications. In *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018. (Pages 1 and 5.)

Rom90. J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd ACM STOC*, pages 387–394. ACM Press, May 1990. (Pages 34 and 41.)

SW05. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In *EUROCRYPT 2005*, *LNCS* 3494, pages 457–473. Springer, Heidelberg, May 2005. (Pages 2 and 10.)

Wee10. H. Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *51st FOCS*, pages 531–540. IEEE Computer Society Press, October 2010. (Page 1.)

Yao86. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. (Page 1.)