Round-Optimal and Communication-Efficient Multiparty Computation

Michele Ciampi[1], Rafail Ostrovsky[2], Hendrik Waldner[1(✉)], and Vassilis Zikas[3]

[1] The University of Edinburgh, Edinburgh, UK
{michele.ciampi,hendrik.waldner}@ed.ac.uk
[2] University of California, Los Angeles, CA, USA
rafail@cs.ucla.edu
[3] Purdue University, West Lafayette, IN, USA
vzikas@cs.purdue.edu

Abstract. Typical approaches for minimizing the round complexity of multiparty computation (MPC) come at the cost of increased communication complexity (CC) or the 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 a 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 protocols that are secure against *malicious* adversaries in the plain model, which we present in this work. Concretely, our two main contributions are:

1) We provide a round-preserving black-box compiler that compiles a wide class of MPC protocols into *circuit-scalable* maliciously secure MPC protocols in the plain model, assuming (succinct) FE combiners.

2) We provide a round-preserving black-box compiler that compiles a wide class of MPC protocols into *circuit-independent*—i.e., with a CC that depends only on the input-output length of the circuit—maliciously secure MPC protocols in the plain model, assuming Multi-Key Fully-Homomorphic Encryption (MFHE). Our constructions are based on a new compiler that turns a wide class of MPC protocols into k-delayed-input function MPC protocols (a notion we introduce), where the function that is being computed is specified only in the k-th round of the protocol.

As immediate corollaries of our two compilers, we derive (1) the first round-optimal and circuit-scalable maliciously secure MPC, and (2) the first round-optimal and circuit-independent maliciously secure MPC in the plain model. The latter MPC achieves the best to-date CC for a round-optimal malicious MPC protocol. In fact, it is even communication-optimal when the output size of the function being evaluated is smaller than its input size (e.g., for boolean functions). All of our results are based on standard polynomial time assumptions.

O. Dunkelman and S. Dziembowski (Eds.): EUROCRYPT 2022, LNCS 13275, pp. 65–95, 2022.
https://doi.org/10.1007/978-3-031-06944-4_3

1 Introduction

Secure multiparty computation (MPC) [23,42] 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[1] [5,12,13,23,24,27–30,37,38,41].

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 [6,21], where the authors present a two-round MPC protocol in the semi-honest model from standard assumptions. Note that the above lower bound holds even when a correlated-randomness setup is assumed. The works [6,9,16,21,34] show that the same bound holds even for maliciously secure MPC, assuming a trusted correlated-randomness setup. However, Garg et al. [18] 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 [4,11,25]. 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. For malicious adversaries, under the assumption that parties have access to correlated randomness, Quach et al. [39] proved that it is possible to design a two-round circuit-scalable MPC protocol that is secure against malicious adversaries under the learning with errors assumption (LWE). Also in the correlated randomness model, Morgan et al. [33] showed that it is possible to construct a two-round circuit-independent[2] two-party computation protocol in which only one party gets the output, by relying only on LWE.[3]

In the case of semi-honest adversaries (without a setup) the works of Ananth et al. [1] and Quach et al. [39] proposed a round-optimal (two-round) circuit-scalable MPC protocol under standard assumptions. Interesting, and most related to our results, Ananth et al. [1] obtained their result by leveraging a connection between round-optimal semi-honest MPC and *functional encryption combiners*. However, their construction does not achieve security against

[1] 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.

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

[3] In the communication model used in [33] in each round only one party can speak. Hence they obtain the best possible security guarantees in such a communication model.

malicious adversaries. The mentioned results raise the following important open question:

> *Is there a round-optimal maliciously MPC protocol secure against dishonest majority*[4] *in the* plain model *based on standard complexity assumptions that achieves* circuit-scalability*, i.e. has a communication complexity that depends only on the depth of the circuit being evaluated and its input and output length?*

As the first of our two main contributions, we answer the above question in the affirmative by extending the investigation of the relation between FE combiners and MPC to the malicious setting. This completes the landscape of circuit-scalable and round-optimal maliciously secure MPC in the plain model. More concretely, we provide a round-preserving black-box compiler that compiles a wide class of MPC protocols into circuit-scalable protocols assuming any *succinct* FE combiner (see below). 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 the above question in the affirmative. Concretely, we propose a round-preserving *black-box compiler* that compiles a wide class of MPC protocols[5] 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 ciphertexts and the size of the description of the encryption and decryption algorithms depend 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 [32], which, in turn, can be instantiated from the LWE assumption [10]. Hence our result yields the first circuit-independent round-optimal malicious MPC in the plain model for a constant number of parties—and therefore specifically to the first two-party-computation protocol—based on standard polynomial-time assumptions. For the case of arbitrary many parties, to our knowledge, compact MFHE is only known to exist based on the Ring-LWE and the Decisional Small Polynomial Ratio (DSPR) assumption [32]. Hence, under these assumptions, we obtain a circuit-independent round-optimal MPC protocol for arbitrary many parties. Deriving compact MFHE for arbitrary many parties—and hence also a circuit-independent round-optimal MPC—from standard polynomial-time assumptions (e.g., LWE) is an interesting open problem.

[4] Unless otherwise specified, all our results are proved secure in the dishonest majority setting.

[5] We require the first 2 rounds of the MPC protocol to be independent from the inputs.

We highlight that all our constructions require the input protocol to achieve a special notion called k-delayed-input function, which we introduce in this work. Informally, in a k-delayed-input function protocol each party has two inputs: 1) a private input (known at the beginning of the protocol) and 2) the function to be computed whose description is needed only to compute the rounds $k, k+1, \ldots$. A k-delayed-input function protocol guarantees that the adversary does not learn more than what it can infer from the evaluation of the function f on the honest parties' input, where f can be adversarially (and adaptively) chosen.

We further show how to turn any MPC protocol that does not require the input to compute the first $k-1$ rounds into a k-delayed-input function protocol.

1.1 Related Work

Functional encryption (FE) [8,35,40] is a primitive that enables fine-grained access control over 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 of the initial FE candidates is secure. Ananth et al. [1] 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 [2] the authors also explore the relation between MFHE and MPC and, among other results, the authors also 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] Cohen et al. [15] proposed a round-optimal *circuit-scalable* MPC protocol which tolerates adaptive corruption (i.e., the identities of the corrupted parties can be decided during the protocol execution). The security of this protocol is proven in the correlated-randomness

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

model under the adaptive LWE assumption and secure erasures (alternatively, sub-exponential indistinguishability obfuscation).

We recall that it is not possible to achieve security with adaptive corruption (with black-box simulation) in the plain model with a constant number of rounds [19]. For this reason, our work focuses on static corruption only.

1.2 Overview of Our Results

In this work we provide two main results which close the gap between communication-efficient and round-optimal maliciously secure MPC. We present two compilers that amplify existing protocols in terms of their communication complexity while preserving their round complexity, which results in the first class of maliciously secure MPC protocols that are communication-efficient and round-optimal.

From FE Combiners to Circuit-Scalable MPC. The first 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 a communication complexity proportional to the depth of the circuit and the length of the input and output of the circuit being evaluated.[7] In summary, we prove the following theorem.

Theorem 1 (informal). *If there exists a 3-delayed-input function ℓ-round MPC protocol Π 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 complexity 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.*

We argue that the four-round protocols proposed in [4,11] can be turned into 3-delayed-input function protocols, which in turn implies that we can obtain a circuit-scalable round optimal MPC protocol from the LWE assumption, since the maliciously-secure four-round OT that the protocol of [11] relies on can also be instantiated using LWE [17]. This allows us to prove the following corollary.

Corollary 1 (informal). *If the LWE assumption holds, then there exists a round optimal MPC protocol that is secure against malicious adversaries whose communication complexity depends only on the security parameter, the depth, the input length and the output length of the circuit being evaluated.*

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

From Circuit-Independent MPC. For the second contribution we show how to combine an MPC protocol with a perfectly 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 to obtain a single ciphertext, which can be decrypted only using all the secret keys. The output of our compiler is a circuit-independent round-optimal MPC protocol that supports $\min\{n_0, n_1\}$ parties where n_0 and n_1 is the number of parties supported by the input MPC protocol and the MFHE scheme respectively. Our second contribution can be summarized as follows.

Theorem 2 (informal). *If there exists a* 2*-delayed-input function* ℓ*-round MPC protocol* Π *that is secure against malicious adversaries which supports* n_0 *number of parties and a perfectly correct, compact MFHE scheme that supports* n_1 *number of parties, then there exists an* ℓ*-round MPC protocol* Π' *that is secure against malicious adversaries whose communication complexity 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 supports* $\min\{n_0, n_1\}$ *number of parties.*

Additionally, it is possible to improve the above result and to obtain a protocol whose communication complexity is only linear in the length of the inputs (and polynomially in the length of the output and the security parameter), by relying on pseudorandom generators (PRGs). 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 a 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 [32] we obtain the following additional corollary.

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

Our treatment advances the state of the art in communication-efficient and round-optimal MPC. Toward this goal, we combine and substantially extend

Table 1. Communication complexity of two-round semi-honest secure and four-round maliciously secure n-party protocols in the plain- and 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^{th} Residuosity, or LWE.

	Communication complexity	Assumptions	Adversarial model	Rounds
[1,39]	$\mathrm{poly}(\lambda, n, d, L_{\mathsf{in}}, L_{\mathsf{out}})$	LWE	Semi-honest	2
[6,21]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	Semi-honest OT	Semi-honest	2
[16]	$\mathrm{poly}(\lambda, n, d, L_{\mathsf{in}}, L_{\mathsf{out}})$	piO and lossy encryption	Semi-honest	2
[20]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	Bilinear Maps	Semi-honest	2
[25]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	QR	Malicious	4
[4]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	DDH/QR/ N^{th} Residuosity	Malicious	4
[11]	$\mathrm{poly}(\lambda, n, \lvert f\rvert)$	Malicious 4-round OT	Malicious	4
[2]	$\mathrm{poly}(\lambda, n, L_{\mathsf{in}}, L_{\mathsf{out}})$	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	Malicious	4
This work*	$\mathrm{poly}(\lambda, n, L_{\mathsf{in}}, L_{\mathsf{out}})$	LWE	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

*Constant number of parties only.

several recent techniques in the literature of FE and MFHE as well as delayed-input MPC. In this section, to assist the reader better navigate through the many technical challenges and details of our result and evaluate its novelty, we review the main technical challenges and our approach to tackling them.

From FE Combiners to Circuit-Scalable MPC. Towards our construction of circuit-scalable MPC, we rely on the recent work of Ananth et al. [1]. In order to build a better intuition for our final solution, we briefly recap their compiler here.

The main building blocks of that 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.

Compiler of Ananth et al. [1]. The construction of Ananth et al. [1] 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 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 a communication complexity of $\mathrm{poly}(\lambda, n, d, L_{\mathsf{in}}, L_{\mathsf{out}})$, 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).

Achieving Malicious Security. Starting from the above approach, we now show how to obtain a circuit-scalable MPC protocol in the case of malicious adversaries (instead of 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.[8] 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 Π, is required to generate the secret key sk_i^f. Hence, if Π 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 not secure, 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

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

is, it achieves *privacy with knowledge of outputs (PKO)* [26,36]. This notion guarantees that the input of the honest parties are protected as in the standard definition of secure MPC, but the output of the honest parties might not be the correct one (e.g., the adversary can force the honest party to output a string of its choice).

Round Preserving Construction: Privacy with Knowledge of Outputs. The first step towards our final construction is to adapt the above idea in such a way that the round complexity of the resulting protocol is kept down to ℓ, while achieving a somewhat reduced security, namely *privacy with knowledge of outputs* [26]. Looking ahead, in the following paragraph, we discuss how to elevate this to full security. For simplicity, we describe our protocol considering only two parties P_0 and P_1 and consider as a building block an MPC protocol Π which consists of ($\ell = 4$)-rounds (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 Π 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, at a high level, as follows (we refer to Fig. 1 for a pictorial representation).

In the first step, the parties run two instances of Blum's coin tossing protocol [7]. In the first instance the party P_0 acts as the sender and in the other instance the party P_1 acts as the sender. In more detail, 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}.[9] 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 Π 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 $\big(x_i, (r_i^0; \rho_i^0, r_i^1; \rho_i^1, r_{1-i}^i, r_i,), (c_1^0, c_1^1, c_2^0, c_2^1)\big)$. 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 $r_0 \oplus r_1$. We highlight that the check that the commitments generated outside of the MPC protocol are generated correctly is not possible in the standard security definition of MPC. To

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

perform these checks we require the underlying MPC to achieve our new notion of k-delayed-input function, which we explain in the end of this section.

Upon receiving the output of g' (evaluated by Π), P_i computes the output running the decryption algorithm of the FE combiner. Using this approach we guarantee that: 1) the ciphertext ct is honestly computed 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^i_{1-i} that P_i receives in the second round corresponds to the value used in the commitment c^i_{1-i} (hence, the master secret key that P_i obtains as part of the output of Π is consistent with the master secret key it has created *outside* of Π).

Unfortunately, we can only prove that the above protocol preserves the privacy of the inputs of the honest parties, but the output computed by 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^f_i and send it to the honest parties. We finally note that it might look like our approach yields to malleability attacks (i.e., the adversary might bias its commitments using honest-parties commitments). Intuitively, such attacks are prevented since we require the adversary to provide the correct opening as part of the input to the MPC protocol. Hence, we delegate to the MPC the prevention of any such malleability attacks.

From PKO to Full Security. The next step is to elevate PKO security to full security. To achieve this, we utilize the PKO-secure to fully-secure compiler of Ishai et al. [26] to turn the above described protocol into a protocol that achieves standard security in a black-box way.

Besides achieving privacy with knowledge of outputs, our protocol also only realizes single-output functionalities instead of multi-output functionalities. In this case, we can also rely on existing compilers to make our protocol supporting multi-output functionalities [3,31].

We note that we can apply those compilers only if they are 1) round-preserving and 2) do not increase the communication complexity by more than a factor of poly(λ). For the sake of completeness we formally argue that this is indeed the case and refer the interested reader to the full version [14].

From to Circuit-Independent MPC. To obtain a circuit-independent MPC protocol, we combine a multi-key fully-homomorphic encryption scheme (MFHE) with a (non-necessarily communication-efficient) MPC protocol Π.

Let us first briefly recall MFHE: A MFHE scheme consists of four algorithms: (1) a setup algorithm Setup that allows for the generation of public-secret key pairs; (2) an encryption algorithm Enc that takes as input a public key and a message and outputs a ciphertext; (3) an evaluation algorithm Eval that takes as input a list of public keys PK, a set of ciphertexts 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; (4) a decryption algorithm Dec that on input all the secret keys, associated

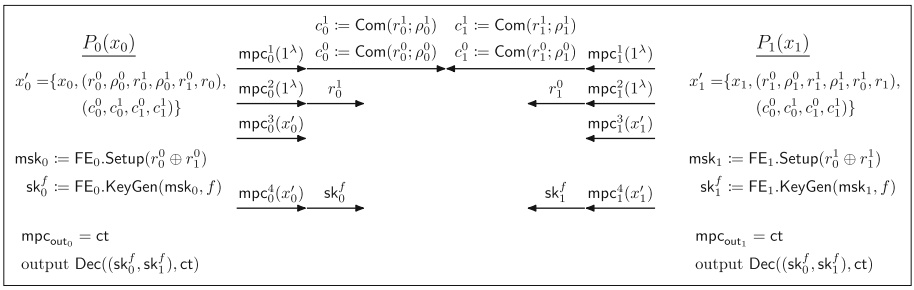

Fig. 1. FE_i, with $i \in \{0,1\}$, denotes a functional encryption candidate. The master secret key for the combiner corresponds to the master secret keys of FE_0 and FE_1. A secret key for the combiner required to evaluate the function f is generated by combining a secret key for FE_0 (sk_0^f) and a secret key for FE_1 (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 $x_i' = \{x_i, (r_i^0, \rho_i^0, r_i^1, \rho_i^1, r_{1-i}^i, r_i), (c_0^0, c_1^0, c_0^1, c_1^1)\}$ 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 $\mathsf{mpc}_{\mathsf{out}_i} = \mathsf{ct}$.

with the public keys of PK, and the ciphertext ct outputs the decryption of ct. Additionally, we require the MFHE scheme to be *compact*, i.e. 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.

Once again, to keep the description simple and to focus on the core ideas, we stick to the two-party case and refer to Sect. 6 for the description of the protocol that supports 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 input 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_1 have input the same list of ciphertexts CT,

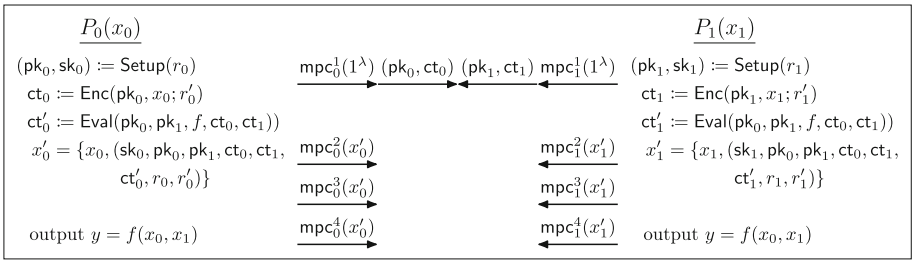

Fig. 2. $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ represents a MFHE scheme. The MPC protocol checks that the cipthertexes ct_0 and ct_1 are in the domain of Enc and that both parties have input the same list of cipthertexes $\mathsf{ct}_0, \mathsf{ct}_1$. Then the MPC protocol decrypts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

2) for each $i \in \{0, 1\}$ it 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. 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 ciphertexts and public keys have been generated correctly and that all the parties have obtained an encryption of the same value when running the MFHE evaluation algorithm. As in the circuit-scalable compiler described before, the check that the public keys and ciphertexts outside of the MPC protocol are generated correctly is not possible in the standard security definition of MPC. To perform these checks we require the underlying MPC protocol to achieve our new notion of k-delayed-input function. The protocol that we have just described is circuit-independent since the size of the public keys and the ciphertexts 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 achieve a communication complexity of $O(L_{\mathsf{in}}) + \mathrm{poly}(\lambda, n, L_{\mathsf{out}})$. To do that, we rely on a folklore technique to reduce the size of the ciphertexts of the MFHE scheme using 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. $\mathsf{Enc}(\mathsf{pk}_i, s_i; r_i^s)$, and sends this encryption along with the value $w_i = \mathsf{PRG}(s_i) \oplus x_i$ to the other party. The size of the resulting message is then $O(L_{\mathsf{in}}) + \mathrm{poly}(\lambda)$. The party P_i, upon receiving $(\mathsf{Enc}(\mathsf{pk}_{1-i}, s_{1-i}; r_{1-i}^s), w_{1-i})$ 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.

k-Delayed-Input Function MPC. As already mentioned in the description of the compilers, we need to rely on an MPC protocol Π that needs the input of the parties only to compute the last two rounds (three in the case of the construction of Fig. 2). Indeed, for the protocol of Fig. 1 for example, the input of each party consists of its actual input, the randomness used to generate its commitments, and *all* the commitments that it has seen (even those generated by the adversary). We note that many existing MPC protocols (e.g., [4,6,11]) indeed do not require the input to compute the first two rounds. However, the fact that the input of the honest parties might be adversarially influenced (e.g., in our protocol some commitments are generated from the adversary) makes it impossible to rely on the standard security notion achieved by such MPC protocols. This is because the standard security notion of MPC requires the inputs 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, and is, therefore, adversarially influenced.

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. That is, these commitments could be thought of as being hardwired in the function evaluated by the MPC protocol Π.

To capture this aspect, we consider a more general notion called k-delayed-input function, where the input of each party consists of two parts, a private input x and a function f. The private part x is known at the beginning of the protocol, whereas the function f does not need to be known before the protocol starts and it is needed only to compute the rounds $k, k+1, \ldots$ of the protocol. We want to guarantee that in the real-world experiment the adversary does not learn more than what it could infer from the output of f, even in the case where it chooses the function f. Equipped with an MPC protocol that satisfies such a definition, we can modify our constructions by letting the parties specify the function that needs to be computed. For example, in the case of the protocol of Fig. 1, the function will contain, in its description, the set of commitments sent in the first round and the messages r_0^1, r_1^2 and uses these values to check that the opening of the commitments are valid with respect to (r_0^1, r_1^2) and only in this case returns a ciphertext for the FE protocol.

To construct a k-delayed-input function protocol, we use a standard $2n$-party ℓ-round MPC protocol Π, where the first $k-1$ rounds can be computed without requiring any input, and a one-time MAC. We refer to the technical part of the paper for more details on how this construction works.

3 Preliminaries

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 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. When it is clear from the context, to not overburden the notation, we do not specify the randomness used in the algorithms unless needed for other purposes.

3.1 Functional Encryption

Definition 3.1 (Functional Encryption [8,35,40]). *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* $\Pr\left[\mathsf{Dec}(\mathsf{sk}_C, \mathsf{Enc}(\mathsf{msk}, x)) = C(x)\right] \geq 1 - \mathrm{negl}(\lambda)$.

In this work, we define the setup algorithm 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.

Definition 3.2 (Single Key Simulation Security of FE [1]). *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 $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, there 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 [1] 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 3.3 (Strong Succinctness). *A functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *for a circuit class* $\mathcal{C}$ *containing circuits* C *that take inputs of length* ℓ_{in} *bits, outputs strings of length* ℓ_{out} *bits and are of depth at most* d *is succinct if the following holds:*

- *The size of the circuit for* $\mathsf{Setup}(1^\lambda)$ *is upper bounded by* $\mathrm{poly}(\lambda, d, \ell_{\mathsf{in}})$ *for some polynomial* poly.
- *Let* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$*, then the size of the circuit for* $\mathsf{Enc}(\mathsf{msk}, \cdot)$ *is upper bounded by* $\mathrm{poly}(\lambda, d, \ell_{\mathsf{in}}, \ell_{\mathsf{out}})$ *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

In this section, we recap the notion of a decomposable functional encryption combiner (DFEC) as introduced by Ananth et al. [1]. In this definition, we rely on the definition of a functional encryption scheme, introduced before (Sect. 3.1).

Definition 3.4 (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* $\mathsf{msk}_i \leftarrow \mathsf{FE}.\mathsf{Setup}_i(1^\lambda)$ *and outputs* $\mathsf{msk} := \{\mathsf{msk}_i\}_{i \in [n]}$*.*

$\mathbf{Real}^{\mathsf{DFEC}}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C, \mathcal{A})$	$\mathbf{Ideal}^{\mathsf{DFEC}}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C, \mathcal{A}, \mathcal{S})$
$\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$	$\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$
$(C_1, \ldots, C_n) = \mathsf{Partition}(n, C)$	$(C_1, \ldots, C_n) = \mathsf{Partition}(n, C)$
$\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, (C_1, \ldots, C_n))$	$\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, (C_1, \ldots, C_n))$
$(I, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}, C)$, 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

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

$\mathsf{KeyGen}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, (C_1, \ldots, C_n))$*: Takes as input the master secret key* msk*, the description of* n *FE candidates* $\{\mathsf{FE}_i\}_{i\in[n]}$ *and a partitioned circuit* $(C_1, \ldots, C_n)$*, and 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*

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

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 introduce the notion of polynomial slowdown.

Definition 3.5 (Polynomial Slowdown [1]). *A decomposable functional encryption combiner* $\mathsf{DFEC} = (\mathsf{Setup}, \mathsf{Partition}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *satisfies 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 3.6 (Single Key Simulation Security of DFEC [1]). *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* $\text{Real}^{\mathsf{DFEC}}$ *and* $\text{Ideal}^{\mathsf{DFEC}}$ *in Fig. 4. A decomposable functional encryption combiner* DFEC *is single key simulation secure, if for any polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *there exists a PPT simulator* $\mathcal{S}$ *and a negligible function* negl *such that:*

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

Definition 3.7 (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* C *that take inputs of length* ℓ_{in} *bits, 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 circuit of* $\mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$ *it holds that* $\mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]}) \leq \text{poly}(\lambda, n, d, \ell_{\mathsf{in}})$.
- *Let* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, \{\mathsf{FE}_i\}_{i\in[n]})$. *For the circuit of* $\mathsf{Enc}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, \cdot)$ *it holds that* $\mathsf{Enc}(\mathsf{msk}, \{\mathsf{FE}_i\}_{i\in[n]}, \cdot) \leq \text{poly}(\lambda, d, \ell_{\mathsf{in}}, \ell_{\mathsf{out}}, n)$ *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* $|\mathsf{aux}| \leq \text{poly}(\lambda, d, \ell_{\mathsf{out}}, n)$ *for some polynomial* poly.

3.3 Multi Key Fully Homomorphic Encryption

Definition 3.8 (Multi-Key Fully Homomorphic Encryption [32]). *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.

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 [32], 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 3.9 (IND-CPA security of MFHE). *A multi-key fully homomorphic encryption scheme* $\mathsf{MFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is secure, if for any PPT adversary* $\mathcal{A}$*, it holds that*

$$\left|\Pr\left[\mathcal{A}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, x_0)) = 1 \,\middle|\, \begin{matrix}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x_0, x_1) \leftarrow \mathcal{A}(\mathsf{pk})\end{matrix}\right]\right.$$
$$\left. - \Pr\left[\mathcal{A}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, x_1)) = 1 \,\middle|\, \begin{matrix}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x_0, x_1) \leftarrow \mathcal{A}(\mathsf{pk})\end{matrix}\right]\right| \leq \mathrm{negl}(\lambda).$$

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 3.10 (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* $|\mathsf{ct}| \leq \mathrm{poly}(\lambda, n)$*, where* $\mathsf{ct} := \mathsf{Eval}(C, (\mathsf{pk}_i, \mathsf{ct}_i)_{i\in[\ell]})$ *with* $\ell \leq n$ *and with the 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. We assume familiarity with the notion of negligible functions, symmetric encryption, digital signatures and commitments and refer to the full version [14] for the formal definitions.

3.4 Secure Multiparty Computation

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. In this work, we denote an ℓ-round MPC protocol as $\pi = (\pi.\mathsf{Next}_1, \ldots, \pi.\mathsf{Next}_\ell, \pi.\mathsf{Out})$, where $\pi.\mathsf{Next}_j$, 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 randomness and the input 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. We assume that readers are familiar with standard simulation-based definitions of secure multiparty computation in the standalone setting. For self-containment we provide the definition in the full version [14] and refer to [22] for a more detailed treatment.

In this work we also consider a relaxed notion of security known as *privacy with knowledge of outputs* [26,36]. 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 3.11 *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:

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

4 *k*-Delayed-Input Function MPC

In this section, we introduce the new notion of *k-Delayed-Input Function*. The classical simulation-based definition of secure MPC requires that the function to be computed is known at the beginning of the real (and ideal) world experiment, before the protocol starts. In our construction we are not in this setting, as we

need an MPC protocol in which the parties can influence the function to be computed by giving an extra input mid-protocol. Concretely, in our protocols, the function computed by the MPC protocol becomes fully defined in the third round (i.e., for the circuit-scalable construction the function incorporates the commitments and the random values sent in the second round).

To capture this, we devise a variant of secure MPC where each party P_i has two inputs x_i and f, where 1) the input x_i is known at the beginning of the real (ideal) world experiment (as in the standard definition of MPC) but 2) the input f can be any function and it becomes known only in the k-th round. In this setting we want to guarantee that if all the honest parties input the same function f, then the adversary either learns the output of f or nothing at all. More formally, we require the input of the honest parties to be protected in the standard simulation based manner for the case where the ideal world evaluates the function f.

A strawman's approach for such a protocol would be to rely on an ℓ-round MPC protocol that does not require the input of the parties to compute the first $k-1$ rounds with $k \leq \ell-1$. We call such protocols *delayed-input* protocols. More precisely, one could consider a delayed-input MPC protocol Π for the universal function g where g takes a pair of inputs from each party P_i denoted with (x, f) and returns $f(x_1, \ldots, x_n)$.

Unfortunately, it is not guaranteed that this approach works since the standard security definition of MPC does not capture the scenario in which an input f for an honest party is chosen adaptively based on the first $k-1$ rounds of the protocol. Therefore, even if all the honest parties follow the naive approach we have just described and use the function f as their input, the adversary might be able compute the output of a function $\tilde{f} \neq f$. It should be noted that the description of the computed function can be part of the output as well, hence, the honest parties will notice that the wrong function has been computed and will reject the output. However, the adversary might have gained much more information from the evaluation of $\tilde{f}$ than it would have gotten by evaluating f.

Syntax & Correctness. Before defining the real and ideal execution, we need to define the syntax of an ℓ-Round k-Delayed-Input Function MPC protocol and its correctness. An ℓ-Round k-Delayed-Input Function MPC protocol is defined as $\Pi = (\mathsf{Next}_1, \ldots, \mathsf{Next}_\ell, \mathsf{Out})$. The next message function Next_1 takes as an input the security parameter in unary form, the input of the party, its randomness and a parameter m that represents the size of the function that will be computed, and returns the first message of the protocol. The next-message function Next_j, with $j \in [k-1]$ takes as input all the messages generated by Π in the rounds $1, \ldots, j-1$ (that we denote with τ_{j-1}) the input and the randomness of P_i and outputs the message $\mathsf{msg}_{j,i}$. The next message function Next_j with $j \in \{k, \ldots, \ell\}$ takes the input of the party P_i, a function f (together with τ_{k-1}) and the randomness of P_i, and returns the message $\mathsf{msg}_{j,i}$. To compute the final output, each party P_i runs Out on input τ_ℓ, its input and randomness. We now define the correctness and the security property that a k-delayed-input function protocol must satisfy.

Definition 4.1 (Perfect Correctness for ℓ-Round k-Delayed-Input Function MPC Protocols). *For any $\lambda, m \in \mathbb{N}$, for any inputs $(x_1, \ldots, x_n) \in (\{0,1\}^\lambda)^n$ and for any set of functions $\{f_\gamma\}_{\gamma \in [n]}$ with $|f_\gamma| = m$ for all $\gamma \in [n]$, it must hold for all $i \in [n]$ that*

- *if $f_1 = \cdots = f_n$ then $\Pr[(\mathsf{Out}(\tau_\ell, x_i, r_i) \neq f(x_1, \ldots, x_n)] = 0$,*
- *if there exists $\alpha, \beta \in [n]$ s.t. $f_\alpha \neq f_\beta$ then $\Pr[(\mathsf{Out}(\tau_\ell, x_i, r_i) \neq \perp] = 0$,*

where $\mathsf{msg}_{1,i} \leftarrow \mathsf{Next}_1(1^\lambda, x_i, m; r_i)$, $\mathsf{msg}_{c,i} \leftarrow \mathsf{Next}_c(\tau_{c-1}, x_i; r_i)$ and $\mathsf{msg}_{j,i} \leftarrow \mathsf{Next}_j(\tau_{j-1}, x_i, f_i; r_i)$ where $r_i \leftarrow \{0,1\}^\lambda$, $c \in \{1, \ldots, k-1\}$ and $j \in [k, \ldots, \ell]$.

We now proceed to defining the security of k-delayed-input function protocols, by describing how the real and the ideal world look like.

The Real Execution. Let us denote $\boldsymbol{x} = (x_1, \ldots, x_n)$ where x_i denotes the input of the party P_i. In the real execution the n-party protocol Π 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 size of the function m, 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 a function f and sends it to the honest parties. Then each honest party P_i uses f to compute the rounds $k, k+1, \ldots, \ell$ 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 $\mathrm{Real}^{\mathsf{DIF\text{-}MPC}}_{\Pi,\mathcal{A}(z),I}(k, \boldsymbol{x})$ 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), the outputs of the uncorrupted parties as well as the function f chosen by the adversary. We let $\mathrm{Real}^{\mathsf{DIF\text{-}MPC}}_{\Pi,\mathcal{A}(z),I}$ denote the distribution ensemble $\{\mathrm{Real}^{\mathsf{DIF\text{-}MPC}}_{\Pi,\mathcal{A}(z),I}(k, \boldsymbol{x})\}_{k \in \mathbb{N}, \langle \boldsymbol{x}, z \rangle \in \{0,1\}^*}$.

The Ideal Execution

- **Send inputs to the trusted party:** Each honest party P_i sends x_i to the ideal functionality. The simulator sends $\{x_j\}_{j \in I}$ and f to the ideal functionality.
- **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 simulator $\mathcal{S}$ and f to P_i for each $i \in [n] \setminus I$.
- **Output of the honest parties:** The simulator $\mathcal{S}$ 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 $\text{Ideal}^{\mathsf{DIF\text{-}MPC}}_{S(z),I}(k, \boldsymbol{x})$ as above. Having defined the real and the ideal world, we now proceed to define our notion of security.

Definition 4.2 (k-Delayed-Input Function MPC). *Let λ be the security parameter. We say that a protocol Π satisfying Definition 4.1 is k-delayed-input function in the presence of malicious adversaries if for every PPT adversary $\mathcal{A}$ attacking in the real world (as defined above) there exists an expected PPT ideal-world adversary $\mathcal{S}$ restricted to query the ideal functionality with the same function f that will appear in the real world experiment output such that for any $I \subset [n]$ the following ensembles are computational indistinguishable*

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

Remark 4.3. We note that Definition 4.2 is very similar to the standard notion of MPC. Indeed, our ideal world can be thought of as the ideal world of the standard definition of MPC for the case where the parties want to evaluate the universal function. We also note that in the ideal world there is no notion of rounds, hence it is not immediately clear how to translate what happens in the real world (where the function f is adaptively chosen in the k-th round by the adversary) into the ideal world (where the ideal world adversary has all the information it needs from the beginning of the experiment). The way we break this asymmetry between the ideal and the real world is exactly by restricting the power of the simulator (i.e., the power of the ideal-world adversary) depending on an event that happens in the real world. In our specific case, we require the admissible simulators (i.e., the admissible ideal world adversaries) to be those that query the ideal world functionality using the same function that will appear in the output of the real world experiment. We note that without this requirement this definition becomes useless since the simulator might query the ideal functionality using a function $\tilde{f}$ that is different from the function f used in the real world, which would allow the simulator to learn more about the honest parties' inputs then it would have by querying the ideal functionality with the function f.

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

Fig. 5. Description of the function g.

From MPC Protocols to k-Delayed-Input Function MPC Protocols. To construct an n party ℓ-round k-Delayed-Input Function MPC protocol

Π^{DIF}, we rely on a $2n$ party ℓ-round MPC protocol Π that does not require the input to compute the first $k-1$ rounds and a one-time MAC scheme $\mathsf{MAC} = (\mathsf{Setup}, \mathsf{Auth}, \mathsf{Verify})$. In our protocol Π^{DIF}, each party P_i controls two parties of Π. One party uses the private input and a MAC key (which is known from the beginning) as its input and the other party uses the function f (received at the end of round $k-1$) authenticated with the MAC key as its input. The MPC protocol Π then checks that the functions are authenticated accordingly to the MAC key and that they are all equal. If this check is successful, Π evaluates the function f over the secret inputs of the parties. Finally, the individual outputs of the function evaluation are returned to one of the two parties of Π controlled by the party P_i. To show that the described protocol Π^{DIF} is indeed k-delayed-input function, we rely on the security of the MPC protocol Π and the unforgeability of the MAC. The security of the MPC protocol Π ensures that the private inputs of the parties are protected and the unforgeability of the MAC is used to enforce that the correct function is used in the protocol execution. Intuitively, if, by contradiction, there exists an adversary that manages to evaluate the function $\tilde{f}$ instead of f then we would be able to construct a reduction to the security of the MAC since the only condition in which Π does not output $\perp$ is the one in which all the parties input the same authenticated function f. If there exists an adversarial strategy that makes Π parse f as $\tilde{f}$, then it must be that $\tilde{f}$ has been authenticated using the MAC key of an honest party. We can extract such a forgery using the simulator of Π (that extracts the input from the parties declared as corrupted).

Now, we describe the construction more formally. Let Π be a $2n$-party MPC protocol that realizes the $2n$-input function g described in Fig. 5 with the property that it needs the input of the parties only to compute the rounds $k, k+1, \ldots, \ell$ with $0 \leq k \leq \ell-1$ where $\ell \in \mathbb{N}$ represents the round complexity of Π. In our k-Delayed-Input Function MPC protocol Π^{DIF}, each party P_i emulates two parties P_i^0 and P_i^1 of Π. Let x_i be the private input of P_i, then P_i performs the following steps.

1. Run Setup to sample a MAC key k_i.
2. Run the party P_i^0 using the input (x_i, k_i) and P_i^1 until the round $k-1$.[10]
3. Upon receiving the function f_i compute $\tau_i \leftarrow \mathsf{Auth}(\mathsf{k}_i, f_i)$ and run P_i^1 using the input (f_i, τ_i).
4. When the protocol Π is finished, P_i outputs the output obtained by P_i^0.

Theorem 4.4. *Let Π be a $2n$-party ℓ-round MPC protocol that securely realizes the function f of Fig. 5 and that requires the input only to compute the rounds $k, k+1, \ldots, \ell$ with $0 \leq k \leq \ell-1$ and let $\mathsf{MAC} = (\mathsf{Setup}, \mathsf{Auth}, \mathsf{Verify})$ be a one-time secure MAC scheme, then the protocol Π^{DIF} described above is an n-party ℓ-round k-Delayed-Input Function MPC protocol.*

The proof for this theorem can be found in the full version [14].

[10] We recall that P_0^i and P_1^i do not need to use the input to compute the first $k-1$ rounds, nonetheless we can specify the input of P_i^1 at the very beginning of the protocol.

5 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 Sect. 2 for a simplified description of the protocol for the two-party case and to Fig. 6 for the formal description of our compiler. Our construction makes use of the following cryptographic tools:

- An ℓ-round k-delayed-input function MPC protocol $\Pi^{\mathsf{M}} = (\Pi^{\mathsf{M}}.\mathsf{Next}_1, \ldots, \Pi^{\mathsf{M}}.\mathsf{Next}_\ell, \Pi^{\mathsf{M}}.\mathsf{Out})$ (not necessarily communication efficient) with $k \geq 3$. In the description of our compiler we assume, without loss of generality, that Π^{M} is 3-delayed-input function.[11]
- 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.

Theorem 5.1. *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_{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 the* ℓ*-round MPC protocol* k*-delayed-input function protocol described in Sect. 4 that realizes* $C^{\mathsf{ct}}_{,\mathsf{Setup},i,R^i_{\mathsf{Setup}}}$ *(Fig. 7), 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 refer to the full version [14] for the formal proof of the theorem.

The following theorem follows immediately from Theorem 5.1 and the definition of strong succinct FE combiners.

Theorem 5.2. *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.*

In the full version [14], we give more details on how our compiler can be instantiated, which leads to the following theorem.

Theorem 5.3. *If the LWE assumption 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 compilers proposed in [3,26,31] we can turn our protocol into one that computes any function under the standard simulation based definition of MPC.

[11] Any k'-delayed-input function MPC with $k' > 3$ can be turned into a 3-delayed-input function MPC protocol since the function received in round 2 can be ignored up to round $k' - 1$.

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

Each $i \in [n]$ party P_i has input $x_i \in \{0,1\}^*$ as its secret input.

Round 1.

1. Sample $r^{i\to i}_{\mathsf{Setup}}, r^{i\to k}_{\mathsf{com}}, r^{\mathsf{Enc}}_i \leftarrow \{0,1\}^\lambda$ for all $k \in [n]$ and set $R^i_{\mathsf{com}} := (r^{i\to k}_{\mathsf{com}})_{k\in[n]}$.
2. Set $x'_i := (x_i, r^{i\to i}_{\mathsf{Setup}}, R^i_{\mathsf{com}}, r^{\mathsf{Enc}}_i)$ and compute $\mathsf{msg}_{1,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Next}_1(1^\lambda, x'_i)$.
3. Sample $r^{i\to k}_{\mathsf{Setup}} \leftarrow \{0,1\}^\lambda$ for all $k \in [n] \setminus \{i\}$, compute $\mathsf{com}^{i\to k}_{\mathsf{Setup}} := \mathsf{Com}(r^{i\to k}_{\mathsf{Setup}}; r^{i\to k}_{\mathsf{com}})$ and set $\mathsf{com}^i_{\mathsf{Setup}} := \{\mathsf{com}^{i\to k}_{\mathsf{Setup}}\}_{k\in[n]}$.
4. Send $(\mathsf{msg}_{1,i}, \mathsf{com}^i_{\mathsf{Setup}})$.

Round 2.

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

Round 3.

1. Let τ_2 denote the transcript of the protocol Π^{M} up to round 2.
2. Compute $\mathsf{msg}_{3,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Next}_3(C^{\mathsf{ct}}_{\mathsf{com}_{\mathsf{Setup},i}, R^i_{\mathsf{Setup}}}, \tau_2)$, with $\mathsf{com}_{\mathsf{Setup},i} := \{\mathsf{com}^k_{\mathsf{Setup}}\}_{k\in[n]}$ and $R^i_{\mathsf{Setup}} := (r^{j\to k}_{\mathsf{Setup}})_{j\in[n],k\in[n]\setminus\{j\}}$.
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 $\mathsf{msg}_{k,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Next}_k(\tau_{k-1})$.
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 $r^{\mathsf{Setup}}_i = \bigoplus_{k\in[n]} r^{k\to i}_{\mathsf{Setup}}$.
3. 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 $\mathsf{sk}_i \leftarrow \mathsf{FE}_i.\mathsf{KeyGen}(\mathsf{msk}_i, C_i; r^{\mathsf{KeyGen}}_i)$ with $r^{\mathsf{KeyGen}}_i \leftarrow \{0,1\}^\lambda$.
4. Compute the fourth round message $\mathsf{msg}_{\ell,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Next}_\ell(\tau_{\ell-1})$.
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 $(\mathsf{ct}, (\tilde{r}^{k\to i}_{\mathsf{Setup}})_{i\in[n],k\in[n]\setminus\{i\}}) \leftarrow \Pi^{\mathsf{M}}.\mathsf{Out}(\tau_\ell)$.
3. Output $\mathsf{DFEC.Dec}(\mathsf{sk}_C, \mathsf{ct})$ with $\mathsf{sk}_C = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.

Fig. 6. Description of the protocol Π^{FE} that securely realizes any functionality with knowledge of outputs.

6 Our Compiler: Circuit-Independent MPC

We now show how to construct a communication efficient MPC protocol that realizes any single-output functionality f. We refer to Sect. 2 for a simplified description of the protocol for the two-party case and to Fig. 8 for the formal description of our compiler We make use of the following tools:

Input: $(x_i, r^{i\to i}_{\mathsf{Setup}}, R^i_{\mathsf{com}}, r^{\mathsf{Enc}}_i)_{i\in[n]}$
- Parse $\mathsf{com}_{\mathsf{Setup},i}$ as $\{\mathsf{com}^k_{\mathsf{Setup}}\}_{k\in[n]}$ and $\mathsf{com}^i_{\mathsf{Setup}}$ as $\{\mathsf{com}^{i\to k}_{\mathsf{Setup}}\}_{k\in[n]}$.
- Parse R^i_{Setup} as $(r^{j\to k}_{\mathsf{Setup}})_{j\in[n],k\in[n]\setminus\{j\}}$.
- Parse R^i_{com} as $(r^{i\to k}_{\mathsf{com}})_{k\in[n]}$ for all $i\in[n]$.
- For all $i,j\in[n]$ check that $\mathsf{com}^{i\to j}_{\mathsf{Setup}} = \mathsf{Com}(r^{i\to j}_{\mathsf{Setup}}; r^{i\to j}_{\mathsf{com}})$.

If one of the above checks fails then output $\perp$, continue as follows otherwise.
For each $i\in[n]$, compute $r^{\mathsf{Setup}}_i = \bigoplus_{k\in[n]} r^{k\to i}_{\mathsf{Setup}}$ and generate $\mathsf{msk}_i \leftarrow \mathsf{FE}_i.\mathsf{Setup}(1^\lambda; r^{\mathsf{Setup}}_i)$
Let $\mathsf{msk} := (\mathsf{msk}_1,\ldots,\mathsf{msk}_n), x = (x_1,\ldots,x_n), r^{\mathsf{Enc}} := \bigoplus_{i\in[n]} r^{\mathsf{Enc}}_i$.
Output: $\mathsf{ct} := \mathsf{DFEC.Enc}(\mathsf{msk}, x; r^{\mathsf{Enc}})$ and $\{r^{k\to j}_{\mathsf{Setup}}\}_{j\in[n],k\in[n]\setminus\{j\}}$ to P_i.

Fig. 7. Circuit $C^{\mathsf{ct}}_{\mathsf{com}_{\mathsf{Setup},i}, R^i_{\mathsf{Setup}}}$.

- An ℓ-round k-delayed-input function MPC protocol $\Pi^{\mathsf{M}} = (\Pi^{\mathsf{M}}.\mathsf{Next}_1,\ldots,\Pi^{\mathsf{M}}.\mathsf{Next}_\ell, \Pi^{\mathsf{M}}.\mathsf{Out})$ (not necessarily communication efficient) with $k\geq 2$.
- A multi-key fully homomorphic encryption scheme $\mathsf{MFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ for n keys.

Theorem 6.1. *Let* MFHE *be a multi-key fully homomorphic encryption scheme with circuit size* cs_{Setup} *for the setup algorithm* MFHE.Setup*, circuit size* cs_{Enc} *for the encryption algorithm* MFHE.Enc*, circuit size* cs_{Dec} *for the decryption algorithm* MFHE.Dec *and ciphertext size* s_{ct}*, let* Π^{M} *be the* ℓ*-round MPC protocol* k*-delayed-input function protocol that realizes the circuit* $C^{\mathsf{Dec}}_{\mathsf{ct}^i,K^i}$ *(Fig. 9), then* Π^{FHE} *is an* ℓ*-round MPC protocol that securely realizes the single-output functionality* C *with communication complexity* $\mathrm{poly}(\lambda, n, cs_{\mathsf{Setup}}, cs_{\mathsf{Enc}}, cs_{\mathsf{Dec}}, s_{\mathsf{ct}})$.

We refer to the full version [14] for the formal proof.

Due to Theorem 6.1 and the definition of a compact multi-key FHE scheme we have the following.

Theorem 6.2. *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.*

We can easily modify Π^{FHE} to obtain a protocol $\Pi^{\mathsf{FHE}'}$ which has a communication complexity of $O(L_{\mathsf{in}}) + \mathrm{poly}(\lambda, n, L_{\mathsf{out}})$. The protocol $\Pi^{\mathsf{FHE}'}$ 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. $\mathsf{Enc}(\mathsf{pk}_i, s_i; r^s_i)$, 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 $(\mathsf{Enc}(\mathsf{pk}_j, s; r^s_j), w_j)$ 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

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

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

Round 1.

1. Sample r_i^{Setup} and $r_i^{\mathsf{Enc}} \leftarrow \{0,1\}^\lambda$.
2. Set $x_i' := (x_i, r_i^{\mathsf{Setup}}, r_i^{\mathsf{Enc}})$ and compute $\mathsf{msg}_{1,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Next}_1(1^\lambda, x_i')$.
3. Compute $(\mathsf{pk}_i, \mathsf{sk}_i) := \mathsf{Setup}(1^\lambda; r_i^{\mathsf{Setup}})$.
4. Compute $\mathsf{ct}_i := \mathsf{Enc}(\mathsf{pk}_i, x_i; r_i^{\mathsf{Enc}})$.
5. 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 $\mathsf{msg}_{2,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Next}_2(C^{\mathsf{Dec}}_{\mathsf{ct}^i, K^i}, \tau_1)$, where $K^i := (\mathsf{pk}_j, \mathsf{ct}_j)_{j\in[n]}$.
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 $\mathsf{msg}_{k,i} \leftarrow \Pi^{\mathsf{M}}.\mathsf{Next}_k(\tau_{k-1})$.
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. 8. The protocol Π^{FHE} that securely realizes f.

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

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

If one of the above checks fails then output $\perp$ else return y to P_i.

Fig. 9. Circuit $C^{\mathsf{Dec}}_{\mathsf{ct}^i, K^i}$

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. 9.

Theorem 6.3. *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.*

Due to [3,31,32] we can claim the following.

Corollary 6.4. *If the LWE and DSPR assumptions hold and any of the DDH, QR,* N^{th} *Residuosity or LWE assumption hold, or there exists a malicious-secure*

OT, then there exists a round optimal (4-round) circuit-independent *MPC protocol that realizes any functionality.*

Acknowledgments. Work done in part while the fourth author was at the University of Edinburgh.

The first author is supported in part 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 HR0011-20-2-0025, NSF grant CNS-2001096, US-Israel BSF grant 2015782, Cisco Research Award, 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, Lockheed-Martin Research Award and Sunday Group. 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 NSF grant no. 2055599 and by Sunday Group.

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

1. Ananth, P., Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: From FE combiners to secure MPC and back. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 199–228. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_9
2. Ananth, P., Jain, A., Jin, Z., Malavolta, G.: Multikey fhe in the plain model. Cryptology ePrint Archive, Report 2020/180 (2020). https://eprint.iacr.org/2020/180
3. Asharov, G., Jain, A., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. Cryptology ePrint Archive, Report 2011/613 (2011). https://eprint.iacr.org/2011/613
4. Badrinarayanan, S., Goyal, V., Jain, A., Kalai, Y.T., Khurana, D., Sahai, A.: Promise zero knowledge and its applications to round optimal MPC. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 459–487. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_16
5. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC, pp. 503–513. ACM Press, May 1990. https://doi.org/10.1145/100216.100287
6. Benhamouda, F., Lin, H.: k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 500–532. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_17
7. Blum, M.: Coin flipping by telephone. In: Gersho, A. (ed.) CRYPTO'81, vol. ECE Report 82–04, pp. 11–15. U.C. Santa Barbara, Dept. of Elec. and Computer Eng. (1981)

8. Boneh, D., Sahai, A., Waters, B.: Functional encryption: definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_16
9. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: optimizing rounds, communication, and computation. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 163–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_6
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012, pp. 309–325. ACM, January 2012. https://doi.org/10.1145/2090236.2090262
11. Rai Choudhuri, A., Ciampi, M., Goyal, V., Jain, A., Ostrovsky, R.: Round optimal secure multiparty computation from minimal assumptions. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 291–319. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64378-2_11
12. Choudhuri, A.R., Ciampi, M., Goyal, V., Jain, A., Ostrovsky, R.: Oblivious transfer from trapdoor permutations in minimal rounds. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part II. LNCS, vol. 13043, pp. 518–549. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90453-1_18
13. Ciampi, M., Ostrovsky, R., Siniscalchi, L., Visconti, I.: Round-optimal secure two-party computation from trapdoor permutations. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 678–710. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_23
14. Ciampi, M., Ostrovsky, R., Waldner, H., Zikas, V.: Round-optimal and communication-efficient multiparty computation. Cryptology ePrint Archive, Report 2020/1437 (2020). https://eprint.iacr.org/2020/1437
15. Cohen, R., Shelat, A., Wichs, D.: Adaptively secure MPC with sublinear communication complexity. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 30–60. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_2
16. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_4
17. Friolo, D., Masny, D., Venturi, D.: A black-box construction of fully-simulatable, round-optimal oblivious transfer from strongly uniform key agreement. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 111–130. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_5
18. Garg, S., Mukherjee, P., Pandey, O., Polychroniadou, A.: The exact round complexity of secure computation. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 448–476. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_16
19. Garg, S., Sahai, A.: Adaptively secure multi-party computation with dishonest majority. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 105–123. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_8
20. Garg, S., Srinivasan, A.: Garbled protocols and two-round MPC from bilinear maps. In: Umans, C. (ed.) 58th FOCS, pp. 588–599. IEEE Computer Society Press, October 2017. https://doi.org/10.1109/FOCS.2017.60

21. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 468–499. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_16
22. Goldreich, O.: The Foundations of Cryptography - Volume 2 Basic Applications. Cambridge University Press, Cambridge (2004)
23. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press, May 1987. https://doi.org/10.1145/28395.28420
24. Goyal, V.: Constant round non-malleable protocols using one way functions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC, pp. 695–704. ACM Press, June 2011. https://doi.org/10.1145/1993636.1993729
25. Halevi, S., Hazay, C., Polychroniadou, A., Venkitasubramaniam, M.: Round-optimal secure multi-party computation. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 488–520. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_17
26. Ishai, Y., Kushilevitz, E., Paskin, A.: Secure multiparty computation with minimal interaction. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 577–594. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_31
27. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_32
28. Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 335–354. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_21
29. Katz, J., Ostrovsky, R., Smith, A.: Round efficiency of multi-party computation with a dishonest majority. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 578–595. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_36
30. Kilian, J.: Founding cryptography on oblivious transfer. In: 20th ACM STOC, pp. 20–31. ACM Press, May 1988. https://doi.org/10.1145/62212.62215
31. Lindell, Y., Pinkas, B.: A proof of security of yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2008). https://doi.org/10.1007/s00145-008-9036-8
32. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Karloff, H.J., Pitassi, T. (eds.) 44th ACM STOC, pp. 1219–1234. ACM Press, May 2012. https://doi.org/10.1145/2213977.2214086
33. Morgan, A., Pass, R., Polychroniadou, A.: Succinct non-interactive secure computation. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 216–245. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_8
34. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_26
35. O'Neill, A.: Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556 (2010). https://eprint.iacr.org/2010/556
36. Paskin-Cherniavsky, A.: Secure computation with minimal interaction. Ph.D. thesis, Computer Science Department, Technion (2012)

37. Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. In: Babai, L. (ed.) 36th ACM STOC, pp. 232–241. ACM Press, June 2004. https://doi.org/10.1145/1007352.1007393
38. Pass, R., Wee, H.: Constant-round non-malleable commitments from sub-exponential one-way functions. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 638–655. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_32
39. Quach, W., Wee, H., Wichs, D.: Laconic function evaluation and applications. In: Thorup, M. (ed.) 59th FOCS, pp. 859–870. IEEE Computer Society Press, October 2018. https://doi.org/10.1109/FOCS.2018.00086
40. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_27
41. Wee, H.: Black-box, round-efficient secure computation via non-malleability amplification. In: 51st FOCS, pp. 531–540. IEEE Computer Society Press, October 2010. https://doi.org/10.1109/FOCS.2010.87
42. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press, October 1986. https://doi.org/10.1109/SFCS.1986.25