Practical and Efficient FHE-based MPC

Nigel P. Smart

KU Leuven, Computer Security and Industrial Cryptography, Leuven, Belgium
Zama Inc., Paris, France

Abstract. We present a *reactive* MPC protocol built from FHE which is *robust* in the presence of active adversaries. In addition the protocol enables reduced bandwidth via means of transciphering, and also enables more expressive/efficient programs via means of a Declassify operation. All sub-components of the protocol can be efficiently realised using existing technology. We prove our protocol secure in the UC framework.

Keywords: Multi-Party Computation · Fully Homomorphic Encryption

Contents

E-mail: nigel.smart@kuleuven.be (Nigel P. Smart)

1 Introduction

Multi-Party Computation (MPC) and Fully Homomorphic Encryption (FHE) are often seen as competing technologies in the cryptographic goal of Computing on Encrypted Data. However, this is a rather naive view. One can base MPC on various different cryptographic building blocks, for example traditionally one based MPC on Linear Secret Sharing Schemes (LSSSs) or Garbled Circuits (GCs). However, one can also base MPC on FHE, as was originally pointed out in Gentry's thesis [Gen09]. Gentry provides a simple Secure-Function Evaluation (SFE) protocol in which parties use an FHE algorithm to encrypt their inputs to each other, the parties then compute independently the function output homomorphically, and then the (public) output is obtained via a distributed decryption protocol.

Since its creation by Gentry, the development of FHE has been rapid. There are now a plethora of schemes to choose from: BGV [BGV12], BFV [Bra12, FV12], CKKS [CKKS17], GSW [GSW13], FHEW [DM15], and TFHE [CGGI20], to name but a few. Each scheme comes with its own advantages and disadvantages, and particular applications for which it is well suited.

One can view the distributed decryption protocol in Gentry's SFE protocol above as a mini-MPC protocol, thus we have (in some sense) built an SFE protocol from an FHE scheme and a mini-MPC protocol, with the key generation itself performed by another mini-MPC protocol (such as that in [RST$^+$22]). There are a few drawbacks with Gentry's blueprint. Firstly, it only provides secure function evaluation (i.e. evaluation of a single function, with a public output, in a one-shot manner), therefore it does not provide full reactive MPC. Secondly, the protocol only provides semi-honest (a.k.a. passive) security. On the other hand the communication complexity is a (admittedly large) function of the input size, i.e. the communication complexity does not depend on the function complexity.

MPC and FHE have been combined in other ways. For example the SPDZ protocol [DPSZ12] produces an actively-secure MPC protocol based on LSSS technology, but one which uses an FHE scheme supporting circuits of multiplicative depth one (so called Somewhat Homomorphic Encryption (SHE) schemes of depth one) as a means of providing an efficient offline phase. Indeed one can view the offline phase of SPDZ as a variant of Gentry's FHE-based MPC protocol for the functionality of producing Beaver multiplication triples. To obtain active security in SPDZ one needs to augment the FHE ciphertexts with Zero-Knowledge Proofs of Knowledge (ZKPoKs). There are ZKPoKs which have been specifically designed for this usage in the SPDZ protocol [BCS19].

In a different direction, [CLO$^+$13] provides a *robust* MPC protocol, in the honest majority setting, which utilizes an SHE scheme of a specified depth, which follows Gentry's blueprint for SFE. The extension from SHE to supporting any function is enabled by replacing the bootstrapping in FHE by a special protocol based on distributed decryption; namely bootstrapping is performed by interaction.

Another combination of MPC and FHE technologies has been the work on Multi-Key FHE (MK-FHE), see [AJL$^+$12, AJW11, LTV12] amongst many other works. In these works, instead of parties generating a global single FHE key in an initialization phase (as in the works above), the parties take their existing individual (multiple) FHE key pairs and combine them in order to perform an MPC-like computation. Whilst this provides a simpler operational setup, the practical implementations of MK-FHE are not as efficient as single key FHE. Another variant of MK-FHE is that of Multi-Party FHE (MP-FHE). The work on MP-FHE is much like our own application; namely there is a distributed key generation protocol and a distributed decryption protocol. However in MP-FHE, the key derivation is created directly from the underlying mathematics, as opposed to applying generic MPC technology. Thus the resulting FHE schemes obtained are slightly different, or have slightly larger parameters, than those proposed in single party FHE. We emphasise that in our work we envisage the use of standard FHE scheme's with the same parameter sets for the single user case; but we use them in a multi-user environment. To distinguish this from MK-FHE and MP-FHE, we call our usage simply Threshold-FHE.

In a seperate line of work, stretching over the last twenty years, a major performance improvement in practical LSSS-based MPC protocols has been seen to come from "opening", or "declassifying" secret information during an MPC computation. This use of Declassify as a basic operation has been exploited in many MPC systems to enable efficient fixed point, floating point and other advanced operations, see for example [Cat21, Cd10, CS10, DFK+06] amongst other works. Whilst the above works on applying FHE to SFE/MPC protocols utilize a method to declassify data, via the distributed decryption protocol inherent in the output operation, they have not exploited the ability to perform distributed decryption as a means of declassifying data during a computation; and thus enabling a richer set of basic operations than just Add and Mult.

In a final line of work, researchers have been investigating transciphering as a means of reducing the huge ciphertexts seen in FHE computations. For example in the above works, the parties need to encrypt via FHE, and so the input (whilst independent of the function size) is still very large (as the FHE ciphertexts are large). In addition an MPC protocol may wish to store data for a long period of time. By transciphering from the FHE cipher to a more compact cipher (for example a simple encryption scheme based on a stream cipher) and back again, one can reduce input bandwidth and storage quite considerably. The paper [DGH+21] considers transciphering for BGV/BFV style FHE schemes, and presents a specialized cipher tuned for these two schemes, called Pasta. They show that with levelled BGV/BFV style FHE such a transciphering, from an encryption under Pasta to an encryption under the FHE scheme, can be done in about 120 seconds for plaintext spaces of 17 and 33 bits. For TFHE style FHE, it would appear that utilizing a standard bit-oriented stream cipher with low multiplicative depth (for example Trivium [De 06]) would be more efficient, or even a small modification of Trivium such as Kreyvium [CCF+16]. In [DGH+21], the authors report on a homomorphic implementation of Kreyvium using TFHE which ouputs one encrypted bit of output every 237 milliseconds. In [BOS23] this is improved to roughly 4 milliseconds per bit.

1.1 Our Contribution

We start by the observation that current FHE schemes are now fast enough that they can be deployed in real life scenarios. In particular bootstrapping is no longer a bottleneck, for example bootstrapping for the TFHE encryption scheme [CGGI20] can be done in under 20 milliseconds [CJL+20]. In addition distributed decryption can be done (depending on the type of FHE scheme, the parameters and the number of parties) in either a few milli-seconds or around half a second [DDE+23][1]. Thus one can reasonably imagine implementing, in the real world, an MPC protocol which utilizes a Threshold-FHE scheme.

Our second observation is that such (practical) protocols need to be fully general and fully secure. By this we mean that the MPC protocol should be *robust*, i.e. it should be maliciously secure and should provide guaranteed output delivery for honest parties, and the input and output parties may be distinct from the parties executing the MPC operation. In addition the protocol should be *composable*, so any security proof needs to be given in the UC framework.

Our third observation is that one wants a protocol that supports fully reactive computation, with both public and private outputs to parties. In particular the "programs" executed by the MPC system should support useful features such as the Declassify operation discussed above.

Our final observation is that one does not want the communication complexity of the input, output and storage of data to depend on the relatively large FHE ciphertexts (and

[1]These are timings using standard processors, if FPGA or ASIC acceleration is applied then these run-times for distributed decryption could be reduced by at least two orders of magnitude.

any associated zero-knowledge proofs which need to accompany them for active security considerations). For example, one may want to support clients entering data which are on a resource bound mobile device. Thus our MPC protocol should support transciphering as a basic primitive.

Thus we provide an FHE based MPC protocol which implements the four main points above. We provide a full UC proof of security for the protocol. We believe this is the first time both transciphering and Declassify operations have been shown to the secure within an FHE-based MPC protocol, as opposed to being secure in a stand-alone manner. We pay particular attention to ensure that *all* building blocks used by our protocol are efficiently implementable with current technology. Thus our protocol can be implemented, and utilized today, to enable low-bandwidth, FHE-based MPC computations.

1.2 System Overview

We work in an MPC model in which we separate the parties into three sets; the parties who provide input $\mathbb{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_I)$, the parties who obtain output $\mathbb{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_O)$, and the parties who perform the computation $\mathbb{C} = (\mathcal{C}_1, \ldots, \mathcal{C}_C)$. This distinguishing of roles between the parties seems to have first been introduced in the FairplayMP system [BNP08], a generalization to secret sharing based MPC was presented in the Cybernetica research report [BK11], with a full presentation being given in [BKL$^+$14].

We assume a monolithic adversary that can *statically* corrupt any number of input parties, any number of output parties, and a specified fraction of the computing parties. The number of computing parties which are corruptible is given by a threshold t, where we assume $t < C/3$ in order to ensure that our distributed decryption protocol is robust[2]. Obviously if all input parties are corrupted then there is little point in performing a secure computation at all, so we can assume that at least one input party is honest.

We present our model in the synchronous communication model, but we believe it can be easily adapted to the asynchronous model (assuming an asynchronous protocol for our Declassify operation, and assuming a standard synchronization point for obtaining parties inputs). Parties are assumed to be connected by authentic channels, except in specific instances where we require private channels (these will be explicitly mentioned when we require them).

1.3 Discussion

Our protocol is essentially an MPC protocol realised via FHE, and not via Yao- or LSSS-based, constructions. We are using this MPC formalisation of an FHE-enabled protocol to capture various use cases of FHE all in one go.

1.3.1 Standard MPC:

In the standard MPC definition we have $\mathbb{I} = \mathbb{O} = \mathbb{C}$. This captures standard MPC use cases of secure data collaboration, and collaborative computing. As already remarked, an initial (passively secure) SFE protocol based on FHE was originally given in Gentry's PhD thesis [Gen09].

The advantage of using our FHE-based MPC protocol, over a standard LSSS- or GC-based MPC protocol, is that the communication costs can be reduced considerably. Indeed our consideration of transciphering within our model is key to reducing communication costs.

In practice network performance improves at a slower rate than computing performance; this is inherent in technology as network latency is bound by the speed-of-light, and network

[2]One can theoretically push this to $t < C/2$, but then the distributed decryption protocol is much less efficient given current technology.

bandwidth is also bound by physical limitations. On the other hand, methods of improving computational efficiency seem to have no limits[3]. Thus it is expected in the long run that computation intensive FHE-based MPC will outperform communication intensive LSSS- or GC-based MPC.

1.3.2 Outsourcing FHE:

In the standard outsourcing scenario, considered in many FHE papers, we have $\mathbb{I} = \mathbb{O} = \{\mathcal{I}\}$, i.e. a single identical input and output party. Then the computation is performed by a single server, $\mathbb{C} = \{\mathcal{C}\}$, with the result returned to the user. That there is only one input party, who must therefore be honest, can simplify the input protocols considered in our paper (as we can assume all inputs are honestly created). Of course some outsourcing examples have that the server also has some private input. In which case one has $\mathbb{I} = \{\mathcal{I}, \mathcal{C}\}$, $\mathbb{O} = \{\mathcal{I}\}$ and $\mathbb{C} = \{\mathcal{C}\}$; in which case one cannot assume that all inputs are validly created.

If we restricted our programs to not have any Declassify instructions then this becomes the standard FHE outsourcing scenario and we can obtain semi-honest security if $\mathbb{C}$ contains just one server[4]. However, to obtain robust security (i.e. to protect against a dishonest server) one needs a form of Verifiable Computation (VC). A simple way to obtain VC, and hence active robust security, is to ensure that $\mathbb{C}$ has at least three parties and the set $\mathbb{C}$ contains an honest majority. This ensures that the function has been computed correctly homomorphically.

The obvious disadvantage of not having a Declassify instruction is that every function needs to be represented as an arithmetic circuit over any underlying ring $\mathcal{R}$, which is a not so expressive a representation. Without the Declassify instruction one is (often) led to more complex representations of the same input program. Thus in practice, for outsourced FHE computation one might still want to use an MPC-like situation where one has multiple servers with at most $t < C/3$ bad servers.

1.3.3 Blockchain Scenario:

In a blockchain scenario one can imagine the parties $\mathbb{I}$ and $\mathbb{O}$ being users of the blockchain, with data stored on the blockchain in encrypted form (when the data is sensitive). The computing parties then become an analogue of network validators, which process a smart contract over this encrypted data. That the input parties must agree on the smart contract in this scenario, implies that the input parties and computing parties automatically have consensus on the program $\mathcal{P}$. The blockchain is simply the data store used by the validators to store the variables and the program. Thus our methodology of transciphering to reduce the cost of data storage will be key to such applications.

In the blockchain scenario one can also imagine a situation in which one actually has two sets of parties making up the computing parties $\mathbb{C}$. The first set $\mathbb{C}^1$ just computes the deterministic operations which require no interaction. We require of these operations that they are executed correctly, thus the consensus mechanism of the blockchain can ensure that the specific deterministic values are computed correctly, i.e. we have that $\mathbb{C}^1$ must contain an honest majority.

When an interactive operation needs to be performed on the FHE ciphertexts, one now passes to a different set of parties $\mathbb{C}^2$ of size C where one can tolerate only $t < C/3$ adversarial parties[5]. In the blockchain situation one can image the set $\mathbb{C}^2$ being the set of blockchain validators.

[3] Despite many people predicting the end of Moore's Law for over two decades, there seems no evidence that the improvement in compute performance is slowing down.

[4] Although we need a slightly different method for the Output command than what we will use, and also a different FHE key setup, for example so that the single party in $\mathbb{O}$ knows the decryption key. These are all minor changes, which simplify the discussion and so we ignore them going forward.

[5] Again $t < C/3$ is chosen as opposed to $t < C/2$ for efficiency reasons with current technology.

Finally, in blockchains it is often important to enable bad actors to be identified, since, for example, they can then be "slashed" in Proof-of-Stake style blockchains. This additional systems security requirement is orthogonal to our work, but is enabled since we utilize robust MPC components which enable the honest parties to detect which adversarial parties are misbehaving.

1.4 Historical Overview

Here we outline some of the prior work, and point out how it relates to our own work.

1.4.1 Trivial FHE-based MPC:

We have already outlined how Gentry's thesis [Gen09], which introduced the first FHE scheme, also explained how a Threshold-FHE scheme could enable a communication efficient, passively secure, SFE solution amongst n parties for a function F. The basic protocol was for the parties to encrypt their inputs using the Threshold-FHE scheme, the parties then evaluate the desired function using the FHE operations, and then the output of the function is obtained via a distributed decryption operation.

That this "trivial" protocol is only passively secure is not due to the computation stage (since for honest majority MPC one can take a majority verdict on the computation, assuming the underlying FHE operations are deterministic in nature). The problem arises from the input and output stages, one requires some way of showing that adversarial parties both produce valid encryptions of their encrypted inputs, and know the underlying messages. Similarly one requires an actively secure threshold decryption protocol. It is this blue-print which we turn into an actively secure protocol in this paper, by using the same methodology to secure the computation phase, but by utilizing zero-knowledge proofs in the input phase, and using the actively secure threshold decryption protocol from [DDE+23].

This "trivial" protocol clearly has communication which does not depend on the "size" $|F|$ of the function being computed. In addition, the number of rounds also does not depend on $|F|$. Indeed, if the distributed decryption protocol round complexity does not depend on the number of parties, then the round complexity is constant. Thus the communication complexity is roughly a linear function of n and the number of function inputs and outputs[6].

In our protocol, by using transciphering and assuming an initialization phase, which can be amortized away, the communication complexity of the input phase, can be exactly that of the (in the clear) inputs. For output of data, one requires distributed decryption to obtain the transciphered outputs in our protocol. Thus whilst the parties accepting output, $\mathbb{O}$, have communication which is exactly that of the size of the actual (in the clear) outputs, the computing parties, $\mathbb{C}$, require communication which is dependent on both n, and the underlying FHE parameters.

1.4.2 Trading Computation for Communication:

Traditional MPC, based on garbled circuits or linear secret sharing, has a high communication complexity which depends on the function F being computed. In GC-based MPC the round complexity is constant, but the communication complexity depends on $|F|$, in LSSS-based MPC both the round and the communication complexity depends on the function F. We remarked above that communication complexity of the trivial FHE-based MPC protocol of Gentry does not depend on the function F, both in terms

[6]We say *roughly* as (depending on the distributed decryption protocol) the parameter sizes of the FHE scheme may depend on n. That it is *linear* in n is due to the fact that almost all distributed decryption protocols will have complexity which is also linear in n.

of communication and round complexity. However, this comes at the expense of needing "expensive" FHE operations (in terms of computational cost).

One can try to trade communication and computation in various ways.

- As already remarked, at a protocol level this is done in the SPDZ protocol [DPSZ12] by (essentially) using a variant of Gentry's protocol as the offline phase to evaluate the function which produces Beaver triples. This offline phase can be built from the BFV or BGV encryption schemes, but it requires addition of (specially designed) zero-knowledge proofs to make it actively secure.
- In [CLO$^+$13] a different protocol level trade-off is performed, where the (relatively expensive) boot-strapping operations inherent in FHE are processed using an interactive (MPC-like) protocol. The approach of Choudhury et al [CLO$^+$13] is justified for FHE schemes, such as BGV and BFV, where bootstrapping is very expensive, but for schemes such as TFHE the work of [CLO$^+$13] makes little sense.

Whilst [DPSZ12] has been implemented and is relatively efficient, we do not know of any implementation of [CLO$^+$13].

In our work we do not make protocol level tradeoffs between communication and computation, instead we design a protocol in which such trade-offs can be made at the "program"/"function" level. The prior referenced work in the MPC domain for large integer, fixed point and floating point operations, e.g. in [Cat21, Cd10, CS10, DFK$^+$06], enable (in any MPC protocol) computation to be traded for communication. In essence they "break" the circuit paradigm of describing functions, by making use of the ability of players to interact in an MPC protocol via opening/declassifying intermediate results. We enable this richer methodology of describing functions to be evaluated via our Declassify operation, which enables trading communication for computation, and we capture this in our programming model by adapting the M-Circuits formalism of [BdO$^+$21].

1.4.3 Threshold-FHE:

As already remarked threshold decryption for SHE/FHE schemes have been used in many prior works. For example the first threshold key generation and decryption algorithm for LWE-based encryption schemes [BD10] was presented at around the same time as Gentry's original thesis. For threshold SHE protocols one can go back to (at least) the SPDZ protocol [DPSZ12], and the already mentioned work of Asharov et al [AJL$^+$12] and Choudhury et al [CLO$^+$13]. Both [AJL$^+$12] and [DPSZ12] are presented in the context of full threshold sharing of the secret, whereas [CLO$^+$13] presents it's results for threshold sharing where $t < n/3$, in the case when $\binom{n}{t}$ is small[7]. That [CLO$^+$13] restricts to small values of $\binom{n}{t}$ is due to its use of Pseudo-Random Secret Sharing (PRSS) to enable Shamir sharing of the noise-flooding term.

Threshold-FHE has been used to construct a generic thresholdizer for arbitrary protocols by Boneh et al. in [BGGK17]. They utilize a methodology of "clearing" out the denominators when using Shamir sharing for large values of $\binom{n}{t}$, which adds complexity to the methodology. The paper [BGG$^+$18] extends this basic construction, by instead using so-called $\{0, 1\}$-LSSS (a form of replicated secret sharing) to avoid the need for "clearing" out the denominators in Shamir sharing. The disadvantage of using $\{0, 1\}$-LSSS is that the size of each parties shares can become exponentially large. Further optimization of these universal thresholdizer's was recently given by Cheon et al [CCK23], using a tree-based secret sharing scheme.

Most works on Threshold-FHE make use of noise flooding to obtain security for their threshold protocols. This approach appears, at first sight, to require a super-polynomial gap between the size of the LWE noise term and the ciphertext modulus. In [BS23] and [CSS$^+$22] a way around such super-polynomial gap, via the Renyi divergence, was found.

[7]Here n is the total number of parties in the underlying secret sharing, and t is an upper bound on the number of adversarial parties amongst those n.

However, this comes at the expense of requiring a security model for distributed decryption which is game-based and not simulation based; this causes a problem when composing such a security model in larger protocols such as ours. In [DDE+23] it is argued that such a super-polynomial gap is irrelevant for FHE schemes such as BGV and BFV, and is only relevant for TFHE. The paper [DDE+23] then goes on to present a methodology for threshold decryption for TFHE which does not rely on the Renyi divergence; instead it uses noise-flooding combined with the efficient bootstrapping procedure inherent in TFHE. In addition [DDE+23] utilizes an online/offline methodology for large values of $\binom{n}{t}$ in order to allow a simple application of Shamir's secret sharing scheme, and thus avoid the use of PRSS in such cases. Hence, [DDE+23] presents a methodology for threshold decryption with a simulation based security proof, and which works for threshold access structures with $t < n/3$.

The application of Threshold-FHE to practical situations has also been investigated in the context of the CKKS FHE scheme (which operates on approximate floating point numbers). For example [ZDH20] examines the case of using CKKS for multiparty Convolution Neural Network applications; giving run-times for various problems in the case of two, four, eight and sixteen parties.

In [MTBH21] a semi-honest MPC protocol is given based on a threshold version of the BFV encryption scheme. They demonstrate the utility of their protocol by giving experimental runtimes of a Private Information Retrieval (PIR) application based on their protocol. One can view the protocol in [MTBH21] as a semi-honest, BFV-specific version of the actively secure, general-FHE protocol considered in our paper.

1.4.4 Multi-Party FHE based MPC:

In [LMK+22] the authors present an efficient manner in which to bootstrap FHEW ciphertexts[8] which are smaller. They then show how to create a full-threshold secret shared version of the FHEW decryption key, by combining existing keys together in the manner used in MP-FHE. They do not detail how the resulting threshold decryption is performed, nor present a performance evaluation.

MP-FHE has also found application in theoretical cryptography. For example [Coh16] constructs constant round protocols for MPC over asynchronous networks, given an asynchronous Byzantine agreement functionality, assuming static malicious adversaries when $t < n/3$. In [CsW19] a two round, semi-honest, SFE protocol is given with sublinear communication complexity.

1.4.5 Multi-Key FHE-based MPC:

The above papers on FHE-based MPC, Threshold-FHE and MP-FHE assume, as we do, a setup algorithm in which an FHE public key is prepared and the computing parties obtain a threshold decryption key share. Such multi-party setup procedures are then implemented using another form of MPC, see for example the one described in [RST+22].

Another approach is to use FHE to enable an ad-hoc form of MPC, which requires no multi-party set up protocol. Instead, it takes existing FHE keys and uses FHE-like operations themselves (in particular a key-homomorphic property enabled by most LWE-based FHE schemes) to produce a shared key which enables further MPC operations. This approach, often dubbed Multi-Key FHE (MK-FHE)[9], was first introduced in concurrent work by Asharov et al [AJW11] and Lopez-Alt et al [LTV12], see also [AJL+12]. Multi-key FHE can be further divided into those schemes which enable a user to encrypt a message in

[8]FHEW is a scheme which is closely related to TFHE.

[9]Although often it is called threshold multi-key FHE, and sometimes also just Threshold-FHE as well. To add further confusion older papers use TFHE to mean "Threshold-FHE", as opposed to the modern usage of TFHE to mean "Torus FHE".

the protocol to a single user key, or those which require encryption to a derived multi-user key.

The papers [AJW11, AJL$^+$12, LTV12] introduce LWE-based FHE schemes, which are key-homomorphic, specifically for the task of creating MK-FHE. The resulting FHE-based MPC protocols are made actively secure using zero-knowledge proofs. The paper of Lopez-Alt et al [LTV12] based it's construction on an NTRU-like assumption, which was further extended in the YASHE FHE scheme [BLLN13]. However, such "overstretched" NTRU-based schemes were subsequently shown to be insecure [ABD16].

The MPC protocols based on MK-FHE are often described in the context of a single cloud which does the actual FHE computations, i.e. one has $C = 1$ in our setting. With the MK-FHE property being used to enable various parties to come together in an ad-hoc manner (in a so called on-the-fly methodology) in order to compute a desired function, using the cloud provider as a service. When $C = 1$ the cloud-based MK-FHE protocols need to require a form of Verifiable Computation (VC) to ensure the FHE part of the operation is carried out correctly. We bypass the need for complex VC operations by assuming an honest majority of computing parties in $\mathbb{C}$, i.e. we assume $t < C/2$ (in fact $t < C/3$).

In this context, of a single cloud provider, the distributed decryption operations need to be performed by the output parties $\mathbb{O}$, and not the computing parties $\mathbb{C}$. A dishonest majority of output parties in these protocols (when executing the threshold protocol to obtain a decryption) can be accommodated (theoretically) by zero-knowledge proofs being applied to the threshold decryption protocol. Our use, by the parties in $\mathbb{C}$, of the threshold decryption proposed in [DDE$^+$23], by passes this need for zero-knowledge proofs, by increasing the number of honest parties in C to a setting in which $t < C/3$.

The key theoretical usage of MK-FHE is to produce very low round MPC protocols. For example if one allows the use of the Common Random String (CRS) model then MK-FHE enables two round actively secure MPC, [MW16]. A similar result [GGHR14] is possible, without using MK-FHE, but using instead Indistinguishability Obfuscation (iO). Assuming MK-FHE is, of course, theoretically nicer than assuming iO, since iO is not so well established a primitive. Two rounds is thus known to be both necessary and sufficient for MPC in the CRS model.

In the plain model (i.e. without a setup assumption such as a CRS) it was for a long time an unknown problem as to how many rounds were both necessary and sufficient for actively secure MPC. In 2016 it was shown [GMPP16] that four rounds were necessary, and then in 2017, using MK-FHE, it was shown [BHP17] that four rounds were indeed sufficient for actively secure MPC. If one assumes a PKI then three rounds are possible, as shown (for semi-malicious adversaries) in [KLP18].

Further, theoretical, work has been carried out on MK-FHE-based MPC protocols. For example, Damgård et al [DPR16] develop a form of *equivocal* FHE, which enables them to extend the work of [AJW11, AJL$^+$12, LTV12] from the static corruption model to the adaptive model, whilst (again) giving a secure MPC protocol in the dishonest majority setting.

Despite over ten years of research very little progress has been made on turning MK-FHE into a practical reality, see for example [BP16, CCS19, CDKS19, CZW17, CM15, KKL$^+$22, KÖA23, KMS22, LP19, PS16, YKHK18]. The extra functionality required by such MK-FHE schemes requires larger parameter sets, and more complex algorithms, than are needed in standard FHE schemes. For example, even the most efficient schemes require FHE scheme parameters which scale with the number of intended users.

Thus multi-key protocols enabling on-the-fly cloud MPC not only require complex zero-knowledge proofs, verifiable computing and such like, they also require the use of FHE technology which is not yet ready for deployment; as it is at least an order of magnitude less efficient than standard (non-multi-key) FHE techniques. Since the interactive set-up assumption for standard Threshold-FHE schemes is a relatively simple, and lightweight, MPC protocol the extra benefit of MK-FHE, whilst theoretically appealing, appears to have no practical realisation or commercial interest.

In our work, since we concentrate only on a setting which in which the primitives that we use are currently ready for deployment, we utilize standard Threshold-FHE as opposed to MK-FHE. However, our setting also enables more expressive MPC programs to be evaluated than the pure arithmetic circuits considered by the above papers on MK-FHE.

1.5 Paper Overview

The required protocol building blocks, namely FHE itself, distributed decryption, transciphering and zero-knowledge proofs, are provided in Section 2. Then in Section 3 we outline the programming model in which our MPC programs will be created. This enables us to describe succinctly the requirements of the system, and also to map them onto different use cases. In Section 4 we present the associated ideal functionality, this maps the programming model of the previous section into a more "cryptographically" compact representation which will be used in our security proof. In Section 5 we present our MPC protocol itself, which builds upon the FHE sub-components introduced in Section 2. Finally, in Section 6 we present a proof that our protocol UC-implements the ideal functionality.

2 Homomorphic Building Blocks

Our protocol relies on same basic building blocks arising from developments in Fully Homomorphic Encryption (FHE); namely FHE itself, Threshold-FHE, FHE transciphering, and ZKPoKs of plaintext knowledge for FHE ciphertexts. We present all these objects generically, with pointers to specific instantiations. All our building blocks can be realised in practical applications today.

2.1 Fully Homomorphic Encryption

We consider an FHE scheme with plaintext space a ring $\mathcal{R}$, which one can think of either as a finite field $\mathbb{F}$ or as a finite ring such as $\mathbb{Z}/(p^k)$ (with $\mathbb{Z}/(2^k)$ being a ring of particular interest). An FHE scheme, with plaintext space a ring $\mathcal{R}$, is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add}, \mathsf{Mult})$ and a space $\mathcal{E}$ of "valid" encryptions. The set $\mathcal{E}$ is a subset of a larger set $\mathbb{E}$. These algorithms have the following signatures:

- $\mathsf{KeyGen}(1^\kappa)$: On input of the security parameter κ, this randomized algorithm produces a public/private key pair $(\mathfrak{pk}, \mathfrak{sk})$.
- $\mathsf{Enc}(m, \mathfrak{pk}; r)$: On input of $m \in \mathcal{R}$, a public key, and randomness from a space of random coins Coins, this produces a ciphertext $\mathfrak{ct} \in \mathcal{E}$.
- $\mathsf{Dec}(\mathfrak{ct}, \mathfrak{sk})$: On input of an element $\mathfrak{ct} \in \mathcal{E}$ and a secret key $\mathfrak{sk}$ this outputs the corresponding plaintext $m \in \mathcal{R}$.
- $\mathsf{Add}(\mathfrak{ct}_1, \mathfrak{ct}_2, \mathfrak{pk})$: On input of two elements $\mathfrak{ct}_1, \mathfrak{ct}_2 \in \mathcal{E}$ which decrypt to $m_1, m_2 \in \mathcal{R}$ this deterministically produces a ciphertext $\mathfrak{ct}_3 \in \mathcal{E}$ which decrypts to $m_1 + m_2$.
- $\mathsf{Mult}(\mathfrak{ct}_1, \mathfrak{ct}_2, \mathfrak{pk})$: On input of two elements $\mathfrak{ct}_1, \mathfrak{ct}_2 \in \mathcal{E}$ which decrypt to $m_1, m_2 \in \mathcal{R}$ this deterministically produces a ciphertext $\mathfrak{ct}_3 \in \mathcal{E}$ which decrypts to $m_1 \cdot m_2$.

The correctness conditions of an FHE scheme are obvious; i.e. every element in $\mathcal{E}$ should be decryptable to the correct value.

To ease notation we write $\mathsf{Add}(\mathfrak{ct}_1, \mathfrak{ct}_2, \mathfrak{pk})$ as $\mathfrak{ct}_1 + \mathfrak{ct}_2$ and $\mathsf{Mult}(\mathfrak{ct}_1, \mathfrak{ct}_2, \mathfrak{pk})$ as $\mathfrak{ct}_1 \cdot \mathfrak{ct}_2$. We also assume that scalars (i.e. non-encrypted values in $\mathcal{R}$) can be added and multiplied into ciphertexts at will. Thus one can form arbitrary arithmetic expressions combining ciphertexts and plaintexts, with the output being a ciphertext if any of the input variables are valid ciphertexts.

A key issue with existing FHE techniques, unlike many traditional public key ciphers, is that it is not possibly to efficiently test whether a supposed ciphertext $\mathfrak{ct}$ lies in $\mathcal{E}$ or in $\mathbb{E}$. In fact the IND-CPA security of the scheme is often reduced to the hard problem of distinguishing elements of $\mathcal{E}$ from elements of $\mathbb{E}$. This causes a problem in protocols as we need to ensure that all ciphertexts which are attempted to be decrypted, are indeed decryptable (i.e. valid ciphertexts). This is, one of the reasons, why we will require zero-knowledge proofs of knowledge (ZKPoKs) below. In addition, not all elements in $\mathcal{E}$ may arise from applications of the Enc function, i.e. $\mathcal{E}$ is not the image of $\mathcal{R}$ under Enc. This is because valid ciphertexts also arise from the application of the homomorphic operations Add and Mult.

There are a plethora of FHE schemes available; each with different properties.

- BGV/BFV: These schemes [Bra12, BGV12, FV12] usually utilize a plaintext ring of $\mathcal{R} = \mathbb{F}_p$, or more generally $\mathcal{R} = \mathbb{F}_p^r$ if r slots are used, where p is a large prime. For such schemes bootstrapping (needed to ensure the output of a series of homomorphic operations lies in $\mathcal{E}$) is very slow.
- CKKS: This scheme [CKKS17] enables a plaintext ring of an approximation to $\mathbb{R}$, i.e. $\mathcal{R} \approx \mathbb{R}$, or an approximation to $\mathbb{R}^r$ if r slots are used. Here bootstrapping is relatively fast, but it only allows a smoothing out of the approximation to the real numbers being used.
- TFHE: This scheme [CGGI20], in its most common implementation, enables a plaintext space of $\mathcal{R} = \mathbb{Z}/(2^k)$ (for a small value of k, say $k = 1$ or $k = 4$). However, bootstrapping is highly efficient, and one can extend the homomorphic operations to arbitrary look up tables on $\mathcal{R}$.

2.1.1 FHE Security:

We assume a slightly different notion of FHE security than perhaps is sometimes used (although this notion was used previously in for example [DPSZ12]); a notion which we dub IND-KEY security.

Definition 1 (IND-KEY)**.** We assume there is a "fake" key generation algorithm denoted $\mathsf{KeyGen}^*(1^\kappa)$, which only outputs public keys, which are indistinguishable from standard public keys. In addition we require that applying the non-fake encryption algorithm with the fake public key, produces a ciphertext which is statistically indistinguishable from an encryption of zero with the fake public key. Thus we assume the following relationships between distributions:

$$\Big\{\mathfrak{pk} : (\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)\Big\} \cong_c \Big\{\widetilde{\mathfrak{pk}} : \widetilde{\mathfrak{pk}} \leftarrow \mathsf{KeyGen}^*(1^\kappa)\Big\}. \tag{1}$$

$$\begin{aligned}\Big\{\mathsf{Enc}(m, \widetilde{\mathfrak{pk}}; r) : \quad \widetilde{\mathfrak{pk}} \leftarrow \mathsf{KeyGen}^*(1^\kappa), \quad m \leftarrow \mathcal{R}, \quad r \leftarrow \mathcal{R}\Big\} \\ \cong_s \Big\{\mathsf{Enc}(0, \widetilde{\mathfrak{pk}}; r) : \widetilde{\mathfrak{pk}} \leftarrow \mathsf{KeyGen}^*(1^\kappa), \quad r \leftarrow \mathcal{R}\Big\}.\end{aligned} \tag{2}$$

A scheme which satisfies equations (1) and (2) is said to be IND-KEY secure.

All the above FHE schemes satisfy this security requirement as their public keys, and ciphertexts, consist of a collection of LWE/Ring-LWE pairs. By the LWE assumption these are indistinguishable from random pairs of elements from the appropriate sets.

FHE security is usually defined via an IND-CPA notion; namely that an adversary, on input of $\mathfrak{pk}$, who selects two messages m_1 and m_2 of his choosing, cannot, on being given a ciphertext $\mathfrak{ct}^*$ encrypting either m_1 or m_2, decide whether $\mathfrak{ct}^*$ encrypts m_1 or m_2. Our IND-KEY notion, however, implies the standard IND-CPA security notion via the following sequence of hybrids: Take the experiment in the standard IND-CPA game in which the encrypted message is the adversarially chosen message m_0, now switch to the using a fake public key $\widetilde{\mathfrak{pk}}$. By our first security assumption (in equation (1)) this hop is indistinguishable. Now hop to encrypting an encryption of zero, again this hop is indistinguishable by our second security assumption (in equation (2)). Now hop to encrypting m_1, and finally hop from the fake public key $\widetilde{\mathfrak{pk}}$ to the real one $\mathfrak{pk}$. These last two hops are indistinguishable for the same reason.

2.2 Threshold-FHE

Formally we define the threshold decryption via two ideal functionalities. The first $\mathcal{F}_{\mathsf{KeyGen}}$, in Figure 1, acts as a set-up assumption for our protocol, needed for the UC proof we provide. It generates a key pair, and secret shares the secret key among the computing parties $\mathbb{C}$ using a linear secret sharing scheme $\langle \cdot \rangle$ which tolerates up to t adversaries. Party $\mathcal{C}_i$'s share of a secret shared value x is denoted by $\langle x \rangle_i$.

One can realise this functionality using a generic MPC protocol. Note, despite wanting active security we do not "complete" adversarial input shares into a complete sharing (as is often done in such situations), as we can assume the implementing MPC protocol for $\mathcal{F}_{\mathsf{KeyGen}}$ does not actually need to do this.

$\mathcal{F}_{\mathsf{KeyGen}}$

```
Init(sid):
    1. Execute $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ for the underlying FHE encryption scheme.
    2. Generate a secret sharing $\langle \mathfrak{sk} \rangle$ of the secret key amongst the $C$ players in $\mathbb{C}$.
    3. Send $(\mathsf{sid}, \mathfrak{pk})$ to all players (including the adversary), and send $(\mathsf{sid}, \langle \mathfrak{sk} \rangle_i)$ to player $\mathcal{C}_i$ (including adversarailly controlled players).
```

Figure 1: The ideal functionality for distributed key generation

The key functionality we want to implement is $\mathcal{F}_{\mathsf{KeyGenDec}}$ given in Figure 2. Note, that this functionality always returns the correct result, irrespective of what the adversary does. A protocol which realises $\mathcal{F}_{\mathsf{KeyGenDec}}$ in the $\mathcal{F}_{\mathsf{KeyGen}}$-hybrid model is given in [DDE+23] when $t < C/3$, for all the FHE schemes above.

$\mathcal{F}_{\mathsf{KeyGenDec}}$

```
Init(sid):
    1. Execute $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ for the underlying FHE encryption scheme.
    2. Send $(\mathsf{sid}, \mathfrak{pk})$ to all players, including the adversary and store the value $\mathfrak{sk}$.
DistDecrypt(sid, ct, $\mathcal{U}$): For a ciphertext $\mathfrak{ct} \in \mathcal{E}$ and a set of players $\mathcal{U}$.
    1. Compute $m \leftarrow \mathsf{Dec}(\mathfrak{ct}, \mathfrak{sk})$.
    2. Output $(\mathsf{sid}, \mathfrak{ct}, m)$ to all players in $\mathcal{U}$, and $(\mathsf{sid}, \mathfrak{ct})$ to the adversary (if the adversary controls no party in $\mathcal{U}$).
```

Figure 2: The ideal functionality for distributed key generation and decryption

The functionality implies that if the target set of players $\mathcal{U}$ in the $\mathsf{DistDecrypt}$ cannot be adversarially corrupted by definition, then the functionality is connected to the players in $\mathcal{U}$ via private channels. When the players in $\mathbb{C}$ implement such a functionality, this implies (for such sets $\mathcal{U}$) that the communication between the sets of players in $\mathbb{C}$ and those in $\mathcal{U}$ is over private channels.

In our protocol the functionality is used in two situations:

- $\mathcal{U}$ contains a single output player, this is when $\mathsf{DistDecrypt}$ is used for private output to an output player. We do not know if the player is adversarially corrupted, thus we need private channels between the functionality and the player in $\mathcal{U}$.
- $\mathcal{U}$ is the set of computing partys $\mathbb{C}$. Here $\mathbb{C}$ could contain an adversarial player, and thus the output can be made in the clear by definition. So we only need authentic channels.

2.3 FHE Transciphering

A big problem with FHE ciphertexts is that they are large, i.e. the bits needed to represent an element in $\mathcal{E}$ is large. Hence, a common proposal to avoid this is to enable a form of homomorphic transciphering via another cipher (usually symmetric). We define the underlying symmetric cipher with the following syntax,

$$\mathsf{Sym}: \begin{array}{ccc} \mathcal{R} \times \mathcal{R}^\rho \times \{0,1\}^* \times \mathbb{N} & \longrightarrow & \mathbb{S}, \\ (m, \mathbf{k}; \mathsf{nonce}, \mathsf{cnt}) & \longmapsto & \mathbf{c} \end{array}$$

Note, the key is an element of $\mathcal{R}^\rho$ as eventually we will be evaluating the symmetric cipher given a homomorphic encryption of the key. We will also require the inverse operation

$$\mathsf{Sym}^{-1}: \begin{array}{ccc} \mathbb{S} \times \mathcal{R}^\rho \times \{0,1\}^* \times \mathbb{N} & \longrightarrow & \mathcal{R}, \\ (\mathbf{c}, \mathbf{k}; \mathsf{nonce}, \mathsf{cnt}) & \longmapsto & m \end{array}$$

with an obvious correctness requirement

$$\mathsf{Sym}^{-1}\Big(\mathsf{Sym}(\ m, \mathbf{k};\ \mathsf{nonce}, \mathsf{cnt}\ ), \mathbf{k};\ \mathsf{nonce}, \mathsf{cnt}\ \Big) = m.$$

In what follows, for simplicity, we assume that $\mathbb{S} \subset \mathcal{R}^\sigma$ for some value of σ (so we can also evaluate Sym^{-1} homomorphically). We also assume that every element of $\mathbb{S}$ corresponds to a valid encryption of some message under Sym[10]. We require the cipher to be IND-CPA secure, which our two constructions below (from a secure PRG and a secure PRF) satisfy.

The cipher Sym is said to be FHE-friendly if there is an efficient way to implement the algorithm homomorphically (with no distributed decryption interactions) within the FHE scheme[11] given a homomorphic encryption of the key, and possibly a homomorphic encryption of the message and/or ciphertext.

Being FHE friendly means that, given a homomorphic encryption of the key

$$\mathsf{Enc}(\mathbf{k}, \mathfrak{pk}) = (\mathfrak{ct}_1, \ldots, \mathfrak{ct}_\rho) = (\ \mathsf{Enc}(k_1, \mathfrak{pk}), \ldots, \mathsf{Enc}(k_\rho, \mathfrak{pk})\ ),$$

and a (plaintext) nonce nonce/counter cnt, one can homomorphically compute

$$\mathsf{Enc}(\ \mathsf{Sym}(\ m, \mathbf{k};\ \mathsf{nonce}, \mathsf{cnt}\ ),\ \mathfrak{pk}\ ) \tag{3}$$

efficiently from $\mathsf{Enc}(\mathbf{k}, \mathfrak{pk})$ and either m or $\mathsf{Enc}(m, \mathfrak{pk})$. In addition one can also compute efficiently

$$\mathsf{Enc}(\ \mathsf{Sym}^{-1}(\ \mathbf{c}, \mathbf{k};\ \mathsf{nonce}, \mathsf{cnt}\ ),\ \mathfrak{pk}\ ) \tag{4}$$

[10] This can be relaxed, but that would require further zero-knowledge proofs, and complications, in our protocol, in order to prove that ciphertexts where actually valid encryptions.

[11] Obviously it also has to be efficiently computable in the clear

efficiently from $\mathsf{Enc}(\mathbf{k}, \mathfrak{pk})$ and either $\mathbf{c}$ or $\mathsf{Enc}(\mathbf{c}, \mathfrak{pk})$.

In particular a vector plaintext message $\mathbf{m}$ can be encrypted, by someone who knows the secret key $\mathbf{k}$, into a compact ciphertext $\mathbf{c}$. Then given $\mathbf{c}$, someone who holds the homomorphically encrypted secret key, $\mathsf{Enc}(\mathbf{k}, \mathfrak{pk})$, can transcipher $\mathbf{c}$ into the encryption of the message $\mathbf{m}$ under the FHE cipher Enc by evaluating equation (4). With access to a distributed decryption functionality one can also transcipher a ciphertext encrypted under Enc to a ciphertext encrypted under Sym, by evaluting equation (3), followed by applying the distributed decryption functionality.

Such a symmetric encryption scheme can be easily derived from an FHE-friendly stream or block cipher;

- If we are given an FHE-friendly stream cipher

$$\mathsf{PRG} : \begin{array}{ccc} \mathcal{R}^\rho \times \{0,1\}^* & \longrightarrow & \mathcal{R}^*, \\ (\mathbf{k}, \mathsf{IV}) & \longmapsto & \mathsf{PRG}(\mathbf{k}, \mathsf{IV}). \end{array}$$

  then we can define Sym by the operation, with $\mathbb{S} = \mathcal{R}$,

$$\mathsf{Sym}(\ m, \mathbf{k};\ \mathsf{nonce}, \mathsf{cnt}\ ) := m + \mathsf{PRG}(\mathbf{k}, \mathsf{nonce})^{(\mathsf{cnt})},$$

  using the nonce as the stream cipher IV, and the counter cnt to index into the resulting keystream.

- If we are given an FHE-friendly block cipher

$$\mathsf{PRF} : \begin{array}{ccc} \mathcal{R}^\rho \times \{0,1\}^* & \longrightarrow & \mathcal{R}, \\ (\mathbf{k}, \mathsf{IV}) & \longmapsto & \mathsf{PRF}(\mathbf{k}, \mathsf{IV}). \end{array}$$

  then we can define Sym by essentially using the block cipher in CTR-mode

$$\mathsf{Sym}(\ m, \mathbf{k};\ \mathsf{nonce}, \mathsf{cnt}\ ) := m + \mathsf{PRF}(\mathbf{k}, \mathsf{nonce} \| \mathsf{cnt}),$$

  using the IV as the nonce concaternated with the counter and $\mathbb{S} = \mathcal{R}$.

Note, the key difference between the block and the stream cipher is that in the block cipher one has a potentially high cost on every invocation, whereas in the stream cipher each invocation (a different increasing value of cnt for the same value of nonce) can be cheap, but the initialization phase (processing the key and nonce) can be expensive.

2.3.1 Usage:

This transciphering can be used (in at least) two ways in an FHE program:

1. To store/read data one can have a global secret key $\mathbf{k}$ which no one knows, but which everyone holds the associated FHE encrypted version (such a key can be obtained by every party entering a key share homomorphically and then summing the keys up to form a new key). To compactly store a homomorphically encrypted ciphertext $\mathfrak{ct}$, encrypting a value m, for long term storage one computes

$$\mathfrak{ct}' \leftarrow \mathsf{Enc}(\ \mathsf{Sym}(\ \mathfrak{ct}, \mathsf{Enc}(\mathbf{k}, \mathfrak{pk});\ \mathsf{nonce}, \mathsf{cnt}\ ),\ \mathfrak{pk}\ )$$

   homomorphically. The value $\mathfrak{ct}'$ is passed to the distributed decryption functionality, which gives us a ciphertext $\mathbf{c} = \mathsf{Sym}(m, \mathbf{k};\ \mathsf{nonce}, \mathsf{cnt})$, which we can then store. Then to read the data $\mathbf{c}$ and return it to the homomorphic domain one computes homomorphically the operation

$$\mathfrak{ct} \leftarrow \mathsf{Enc}(\ \mathsf{Sym}^{-1}(\ \mathbf{c}, \mathsf{Enc}(\mathbf{k}, \mathfrak{pk});\ \mathsf{nonce}, \mathsf{cnt}\ ),\ \mathfrak{pk}\ )$$

2. For efficient input and output of data from parties, each party $\mathcal{P}_i$ can enter a key $\mathbf{k}_i$ for the PRG/stream cipher PRG into the system by inputing the homomorphically

encrypted key $\mathsf{Enc}(\mathbf{k}_i, \mathfrak{pk})$. Then to enter some data $m \in \mathcal{R}$ the user actually inputs the clear value $\mathbf{c} = \mathsf{Sym}(\ m, \mathbf{k}_i;\ \mathsf{nonce}, \mathsf{cnt}\ )$, for some unique value $(\mathsf{nonce}, \mathsf{cnt})$. Then the system computes the homomorphic encryption of m homomorphically as above. For sending data, stored in the FHE ciphertext $\mathfrak{ct}$, back to the user, one computes the inverse operation, as above, using the distributed decryption functionality.

Note, both of these programmatic "optimizations" are supported by our F-Circuit formalism below. Whether one utilizes them in a program is up to the user, and we do not discuss their further use.

2.4 Zero-Knowledge Proofs of Plaintext Knowledge

The zero-knowledge proofs we use will always be called with a prover being a party in $\mathbb{I} \cup \mathbb{O} \cup \mathbb{C}$ and the verifiers being the set $\mathbb{C}$. We will apply the ZKPoKs to prove the specific NP-relation R_{enc} given by

$$R_{\mathsf{enc}} = \Big\{ (\mathfrak{ct}, (m, r))\ :\ m \in \mathcal{R},\quad r \in \mathsf{Coins},\quad \mathfrak{ct} = \mathsf{Enc}(m, \mathfrak{pk}; r) \Big\}.$$

The zero-knowledge proofs will be defined by four algorithms, which make use of a random oracle:

- $\mathsf{Prv}(\mathfrak{ct}, (m, r))$: A prover algorithm which, on input of a ciphertext $\mathfrak{ct}$ and a witness (m, r) for the relation R_{enc}, will produce a proof π.
- $\mathsf{Ver}(\mathfrak{ct}, \pi)$: A verifier algorithm which, on input of a ciphertext $\mathfrak{ct}$ and a proof, will output true or false.
- $\mathsf{Sim}(\mathfrak{ct})$: A simulator algorithm which, on input of a ciphertext $\mathfrak{ct}$, will produce a simulated proof π; it does so by programming the underlying random oracle.
- $\mathsf{Ext}(\mathfrak{ct})$: This knowledge extraction algorithm, which has black box (potentially rewindable) access to the proving algorithm will extract the underlying witness for the ciphertext $\mathfrak{ct}$ from a prover which holds this witness.

For all of the FHE schemes mentioned earlier (BGV,BFV,CKKS and TFHE) the encryption algorithm Enc is of the following form: It generates some "small" random integer values (which one can consider as sums of bits, where the bits are part of the random coins in r), then two values $(\mathbf{a}, \mathbf{b})$ are constructed which are *linear combinations* of the random bits, the message, and the values in the public key $\mathfrak{pk}$. The final ciphertext is the pair $(\mathbf{a}, \mathbf{b})$. One can therefore interpret the NP-relation R_{enc} as a (multiple) subset-sum relation over the hidden bits in the random coins r and the bits making up the message m.

Many proof systems for subset-sum/lattice like relations exhibit a form of *soundness slack*. This means that the statement, which we can guarantee a dishonest prover actually commits to, is a slightly modified version of the one which an honest prover is using. Use of such proof systems in FHE-based protocols result in us having to increase parameters in order to deal with the *slack* introduced by such proofs[12]. To ensure simpler protocols, and optimal parameters for our underlying FHE scheme, we require our zero-knowledge proofs to exhibit no such soundness slack. With this requirement in mind there appears two forms of (efficient) zero-knowledge proofs one could use.

2.4.1 Proofs Based on MPC-in-the-Head:

If post-quantum secure proofs are required then one technique to prove such subset sum NP-relations is to use MPC-in-the-Head based zero-knowledge proofs. For example the techniques of [FMRV22] and [FR22] can be applied directly to this situation resulting in a very simple proof technique. Both these papers are based on specializations of the general KKW [KKW18] method for MPC-in-the-Head. At heart these are interactive proof

[12]Effectively the FHE noise in fresh ciphertexts is increased, resulting in larger parameters, or more bootstrapping, or both.

systems (essentially Σ-protocols), but they are made non-interactive using the Fiat–Shamir heuristic in the standard way. In these proofs the simulation algorithm Sim works by programming the random oracle challenges, and the extraction algorithm Ext works (again by programming the random oracle) by rewinding the prover and issueing the prover different responses to it's random oracle queries.

2.4.2 Proofs Based on Vector Commitments:

If one is prepared to accept pre-quantum security assumptions (in particular hardness assumptions in elliptic curve groups which support pairings) one can utilize proofs based on a vector commitment scheme, [Lib23]. These proofs are proven secure in the Algebraic Group Model (AGM) [FKL18]. The prover, verifier and simulator work roughly (from a very high level) just as in the previous case; with the simulator also needing to program the underlying random oracle. However, the extraction algorithm works in the AGM and does not need to rewind the prover. The extraction algorithm simply "observes" the group operations performed by the prover, and from these is able to extract the underlying witness.

In both cases the knowledge extraction via rewinding or via the use of the AGM, causes the underlying ZKPoK to not be UC secure. However, this does not cause an issue in our main protocol security proof. We can still prove UC security of our underlying MPC-FHE protocol. The reason for this is as follows: Our UC protocol simulator will know the secret key of the underlying cryptosystem and so can extract the important part of the witness, i.e. m, by simply decrypting. In our security proof that the simulator is valid, where we do not know the secret key, we can extract the witness at that point by either rewinding the environment or using the AGM. In particular the rewinding/AGM execution is required only for the security reduction that the protocol simulator is correct, and not for actually implementing the protocol simulator. Thus the proof will ensure we have a UC-secure protocol.

3 Basic Multi-Party Programming Model

We wish the parties to be able to perform a reactive computation, i.e. the function to be computed can depend on any publicly revealed values of the computation. The computation is performed over the ring $\mathcal{R}$ supported by the FHE scheme.

3.1 The Programming Model

We present the functions to be computed not as arithmetic circuits over $\mathcal{R}$, but as an "extended" arithmetic circuit, i.e. an arithmetic circuit which can contain additional gadgets/instructions. This mirrors how MPC systems are built in the real world, and was formalized by the concept of M-Circuit from [BdO+21]. We adopt the notion of M-Circuit and slightly adapt the definitions in what follows to the special case of FHE operation. One can see our definitions as a specialisation/extension of the general M-Circuit concept presented in [BdO+21]. As we extend the type system of M-Circuits to cope with FHE-based transciphering, we call our notion an F-Circuit to avoid confusion with the more MPC-inspired M-Circuits from [BdO+21].

Our machine state state consists of a set of registers $\mathbb{V} = \{\mathsf{varid}_i\}$. Each register has a *fixed* type, which is one of the following four types

- $(\mathcal{R}, -)$: This refers to a variable which holds a non-sensitive value in the ring $\mathcal{R}$.
- $(\mathcal{R}, s)$: This refers to a variable which holds a *sensitive* value in the ring $\mathcal{R}$, it equates (in our protocol) to a variable which is stored under the FHE encryption scheme Enc.

- $(\mathcal{R}, t)$: This again refers to a variable which holds a sensitive value in the ring $\mathcal{R}$, it equates (in our protocol) to a variable which has been *transciphered* into an encryption under the scheme Sym for storage purposes.
- $(-, -)$: This refers to a variable which holds a non-sensitive integer value (in some bounded range, say $[0, \ldots, 2^{64}]$).

There is a special integer register pc of type $(-, -)$. The register pc can be alterred by a program, and will be used as the program counter.

One can extend the state machine to include stacks of the above register types (and not just flat register banks), as well as memory banks which can be indexed by values of type $(-, -)$. It is also convenient if the stack values can be indexed by relative indexes of type $(-, -)$. These extensions are immediate and left to the reader. The key point is that a register, stack, or memory bank has a fixed type which does not change during computation. Thus "casting" needs to be done via assignment. We refer to the fixed type of a register varid by the notation $\mathsf{type}(\mathsf{varid}) = (\mathsf{type}^1(\mathsf{varid}), \mathsf{type}^2(\mathsf{varid}))$.

An instruction

$$\mathsf{instr} \in \{\mathsf{Input}, \mathsf{Output}, \mathsf{Add}, \mathsf{Mult}, \mathsf{Declassify}, \mathsf{Trans}^{s \to t}, \mathsf{Trans}^{t \to s}, \mathsf{Terminate}\}$$

operates on the machine state as follows:

- $\mathsf{Input}(\mathsf{varid}, i)$: This accepts an input (of the type associated with varid) from player $\mathcal{I}_i$ for $i \in [1, \ldots, I]$. We require $\mathsf{type}^2(\mathsf{varid}) \in \{s, -\}$, where if $\mathsf{type}^2(\mathsf{varid}) = s$ then it is assumed that this is private input from player $\mathcal{I}_i$.
- $\mathsf{Output}(\mathsf{varid}, i)$: This outputs the value of varid to player $\mathcal{O}_i$ for $i \in [1, \ldots, O]$. We require $\mathsf{type}^2(\mathsf{varid}) \in \{s, -\}$, and if $\mathsf{type}^2(\mathsf{varid}) = s$ then it is assumed that player $\mathcal{O}_i$ learns the clear value of varid, even though it is not stored in clear form.
- $\mathsf{Add}(\mathsf{varid}_z, \mathsf{varid}_x, \mathsf{varid}_y)$: This computes $\mathsf{varid}_z = \mathsf{varid}_x + \mathsf{varid}_y$ where $\mathsf{type}^2(\mathsf{varid}_x)$, $\mathsf{type}^2(\mathsf{varid}_y) \in \{-, s\}$. We require $\mathsf{type}^2(\mathsf{varid}_z) = s$ if either $\mathsf{type}^2(\mathsf{varid}_x)$ or $\mathsf{type}^2(\mathsf{varid}_y)$ are equal to s, otherwise $\mathsf{type}^2(\mathsf{varid}_x) = -$. We also require that $\mathsf{type}^1(\mathsf{varid}_x) = \mathsf{type}^1(\mathsf{varid}_y) = \mathsf{type}^1(\mathsf{varid}_z)$, although see the note about implicit coercion if varid_x or varid_y is a non-sensitive value in $\{0, 1\} \subset \mathbb{Z}$ or $\{0, 1\} \subset \mathcal{R}$.
- $\mathsf{Mult}(\mathsf{varid}_z, \mathsf{varid}_x, \mathsf{varid}_y)$: This computes $\mathsf{varid}_z = \mathsf{varid}_x \cdot \mathsf{varid}_y$ where (again) $\mathsf{type}^2(\mathsf{varid}_x)$, $\mathsf{type}^2(\mathsf{varid}_y) \in \{-, s\}$. We, again, require $\mathsf{type}^2(\mathsf{varid}_z) = s$ if either $\mathsf{type}^2(\mathsf{varid}_x)$ or $\mathsf{type}^2(\mathsf{varid}_y)$ are equal to s, otherwise $\mathsf{type}^2(\mathsf{varid}_x) = -$, and again require that $\mathsf{type}^1(\mathsf{varid}_x) = \mathsf{type}^1(\mathsf{varid}_y) = \mathsf{type}^1(\mathsf{varid}_z)$. Again, see the note about implicit coercion if varid_x or varid_y is a non-sensitive value in $\{0, 1\} \subset \mathbb{Z}$ or $\{0, 1\} \subset \mathcal{R}$.
- $\mathsf{Declassify}(\mathsf{varid}_y, \mathsf{varid}_x)$: This takes an input register of type $\mathsf{type}(\mathsf{varid}_x) = (\mathcal{R}, s)$ and produces an output register of type $\mathsf{type}(\mathsf{varid}_y) = (\mathcal{R}, -)$, which holds the same value. *Note, in the protocol this requires interaction.*
- $\mathsf{Trans}^{s \to t}(\mathsf{varid}_y, \mathsf{varid}_x)$: This performs the identity operation on the underlying value. The input type $\mathsf{type}(\mathsf{varid}_x)$ is of type $(\mathcal{R}, s)$, and the output type $\mathsf{type}(\mathsf{varid}_y)$ is of type $(\mathcal{R}, t)$. *Note, in the protocol this operation requires interaction.*
- $\mathsf{Trans}^{t \to s}(\mathsf{varid}_y, \mathsf{varid}_x)$: This performs the identity operation on the underlying value. The input type $\mathsf{type}(\mathsf{varid}_x)$ is of type $(\mathcal{R}, t)$. and the output type $\mathsf{type}(\mathsf{varid}_y)$ is of type $(\mathcal{R}, s)$. *Note, in the protocol this operation requires no interaction.*
- $\mathsf{Terminate}$: This simply stops the computation and deletes the internal state, i.e. the contents of all registers.

To ease notation we can write a sequence of addition and multiplication instructions as an arithmetic expression, e.g.

$$\mathsf{varid}_z \leftarrow \mathsf{varid}_a \cdot \mathsf{varid}_x + \mathsf{varid}_b \cdot \mathsf{varid}_y + \mathsf{varid}_c.$$

We assume an implicit coercion between the values $\{0, 1\} \in \mathcal{R}$ and the values $\{0, 1\} \in \mathbb{Z}$, for the purposes of addition and multiplication[13]. Thus one can form expressions of the

[13]We could extend coercion to any subset of $\mathbb{Z}$ which could be represented as values in $\mathcal{R}$. For example

form

$$a \leftarrow b \cdot c + (1 - b) \cdot d$$

where a, c and d have type $(-, -)$ and b is of type $(\mathcal{R}, -)$ but it is guaranteed that $b \in \{0, 1\}$. Note, one cannot compute arithmetic operations on items of $\mathsf{type}^2(\cdot) = t$.

A program $\mathcal{P}$ is an ordered sequence of instructions, $\mathcal{P} = [\mathsf{instr}_1, \ldots, \mathsf{instr}_l]$. We execute the program by setting $\mathsf{pc} \leftarrow 1$. to start with and then executing the following steps in turn.

- Fetch the instruction instr indexed by pc.
- Increment pc by one.
- Execute the operation defined by instr.

Note, conditional branching on non-sensitive values is allowed via assigning to the pc value via

$$\mathsf{pc} \leftarrow \mathsf{varid}_b \cdot 100 + (1 - \mathsf{varid}_b) \cdot 200$$

which will jump to instruction 100 if $\mathsf{varid}_b = 1$ and jump to instruction 200 if $\mathsf{varid}_b = 0$.

Subroutine execution can also be supported by using the clear variables to maintain a call-return stack for the pc counter; with a call resulting with a push to the stack followed by a pc assignment, and a return resulting in a pop from the stack and then a pc assignment. Variable passing can be performed using stacks for other data types in the standard manner. Local variables (for subroutines) can be dealt with via the usual stack handling mechanisms.

Different FHE schemes may allow different gadgets/instructions; for example TFHE allows the homomorphic evaluation of arbitrary look up tables (LUTs) over $\mathcal{R}$. Such LUTs can be expressed (inefficiently) in our F-Circuit model, as every lookup table is effectively a (high-degree) polynomial evaluation. Thus one can "encode" a program which utilizes LUTs in the above F-Circuit formalism, and then, when executing it, one can utilize the more efficient implementation enabled in TFHE.

On the other hand BGV/BFV/CKKS allow operations on vectors and associated homomorphic slot-wise manipulation operations. Again one can see these as simply more compact ways to encode the register manipulations we present in an F-Circuit description.

One can run BGV and BFV in a "levelled mode" only, in which no bootstrapping is allowed. In such a situation one needs to restrict the class of programs $\mathcal{P}$ that our protocol will support, so that no value will exceed the maximum level. To do this one needs to augment the type of a variable varid with its "level" if it is of type $(\mathcal{R}, s)$. Then the operations on elements of type $(\mathcal{R}, s)$ need to keep track of the associated level, returning a failure if the level bound is exceeded. Such an adaption of our model is easy, but tedious, and obviously less suitable for long term reactive computations so we leave this extension to the reader. A different way around the problem of failing if the level bound is exceeded, is to use the $\mathsf{Declassify}$ operation to perform a type of interactive bootstrapping; this is essentially the main idea behind the protocol in [CLO+13].

3.2 Compilation

A real program is going to be written in a high level language, this high level language representation needs to be compiled down to the representation given above. This compilation process is itself a security concern. As discussed in [BdO+21] for M-Circuits, two functionally equivalent compilations of the same input program can be such that one leaks private data and one does not. In this paper we assume that a suitable security analysis has been applied to the compilation process, and that any revealed values (for example in $\mathsf{Declassify}$ operations) are proved not to reveal sensitive information. Thus this paper considers the security issues related to executing the F-Circuit representation of the function, on the assumption that the F-Circuit does exactly what the users expect it to.

if $\mathcal{R} = \mathbb{F}_p$, then one could consider coercion between the whole of $\mathcal{R}$ and the subset $\{0, \ldots, p-1\} \subset \mathbb{Z}$.

The program to be computed $\mathcal{P}$ must be agreed (by consensus) by the input $\mathbb{I}$ and computing parties $\mathbb{C}$. This is particularly important as the program could simply be of the form

```
Input(x, 1)
Declassify(y, x)
```

which would output player $\mathcal{I}_1$'s private input.

Our security requirement is the standard robust MPC requirement that the adversary only learns data which is revealed to it via the program (i.e. any declassified values, and values which it enters as inputs, or receives as output, and any values which can be deduced from that), in addition if the adversary deviates from the program then the honest parties will still compute the values as required. This is captured in an ideal functionality given in Section 4.

To obtain robustness we will require $t < C/3$, mainly to enable the efficient distributed decryption operation, given in [DDE+23], to be used in the Declassify operation. This could be relaxed to $t < C/2$ (theoretically) if a more complicated Declassify operation was created.

4 Ideal Functionality

The ideal functionality, in Figure 3 below, specifies exactly what we would like an ideal execution of a F-Circuit amongst the parties to provide. Again, we re-iterate that we make no claim that the execution of the F-Circuit provides the desired application security concerns (that is a problem for the compiler), we are only interested in defining abstractly what the execution of the F-Circuit should enable.

Again we assume a register bank $\mathbb{V} = \{\mathsf{varid}_i\}$, which again (for simplicity) we assume is a flat single file of registers, i.e. no stacks or index-able memory. The ideal functionality keeps track of registers which are in the clear, as well as registers which are meant to be held securely within the functionality.

For simplicity of exposition we allow this ideal functionality to output an abort, but this only happens for a badly typed program; which should be caught by the compiler. Thus the reader should treat the abort's as purely syntactic sugar for their reading, and not as actual abort's by the functionality.

Note the ideal functionality assumes that all players agree on each instruction sent to it, thus this implies consensus on the input F-Circuit program.

$\mathcal{F}_{F-Circuit}$

Init: On input $(\mathsf{sid}, \mathsf{Init}, \mathcal{R})$ from all parties store $\mathcal{R}$.

Input: On input of $(\mathsf{sid}, \mathsf{Input}, \mathcal{I}_i, \mathsf{varid}, x)$, for $\mathcal{I}_i \in \mathbb{I}$, from $\mathcal{I}_i$ and $(\mathsf{sid}, \mathsf{Input}, \mathcal{I}_i, \mathsf{varid})$ from all other parties:

1. If $\mathsf{type}^2(\mathsf{varid}) \notin \{s, -\}$ then abort.
2. If $\mathsf{type}^1(\mathsf{varid}) = \mathcal{R}$ and $x \in \mathcal{R}$ then store x in varid.
3. If $\mathsf{type}^1(\mathsf{varid}) = -$ and $x \in \mathbb{Z}$ then store x in varid.
4. If $\mathsf{type}^2(\mathsf{varid}) = -$ then send x to the adversary.
5. If $x = \perp$ then store the value zero in varid.
   This last case corresponds to an adversarial input x which is mistyped.

Output: On input of $(\mathsf{sid}, \mathsf{Output}, \mathcal{O}_i, \mathsf{varid})$ for $\mathcal{O}_i \in \mathbb{I}$ from all parties:

1. If $\mathsf{type}^2(\mathsf{varid}) \notin \{s, -\}$ then abort.
2. Send the value x stored in varid to $\mathcal{O}_i$.
3. If $\mathsf{type}^2(\mathsf{varid}) = -$ then also send x to the adversary.

Add: On input of $(\mathsf{sid}, \mathsf{Add}, \mathsf{varid}_z, \mathsf{varid}_x, \mathsf{varid}_y)$ from all parties:

1. If $\mathsf{type}^1(\mathsf{varid}_x) \neq \mathsf{type}^1(\mathsf{varid}_y)$ or $\mathsf{type}^1(\mathsf{varid}_x) \neq \mathsf{type}^1(\mathsf{varid}_z)$ then abort.
   With a caveat here in relation to the implicit coercions mentioned earlier.
2. If $\mathsf{type}^2(\mathsf{varid}_x) \notin \{s, -\}$ then abort.
3. If $(\mathsf{type}^2(\mathsf{varid}_x) = s$ or $\mathsf{type}^2(\mathsf{varid}_y) = s)$ and $\mathsf{type}^2(\mathsf{varid}_z) \neq s$ then abort.
4. Retrieve x from varid_x and y from varid_y and store $z = x + y$ in varid_z.

Mult: On input of $(\mathsf{sid}, \mathsf{Mult}, \mathsf{varid}_z, \mathsf{varid}_x, \mathsf{varid}_y)$ from all parties:

1. If $\mathsf{type}^1(\mathsf{varid}_x) \neq \mathsf{type}^1(\mathsf{varid}_y)$ or $\mathsf{type}^1(\mathsf{varid}_x) \neq \mathsf{type}^1(\mathsf{varid}_z)$ then abort.
   Again, with a caveat here in relation to the implicit coercions mentioned earlier.
2. If $\mathsf{type}^2(\mathsf{varid}_x) \notin \{s, -\}$ then abort.
3. If $(\mathsf{type}^2(\mathsf{varid}_x) = s$ or $\mathsf{type}^2(\mathsf{varid}_y) = s)$ and $\mathsf{type}^2(\mathsf{varid}_z) \neq s$ then abort.
4. Retrieve x from varid_x and y from varid_y and store $z = x \cdot y$ in varid_z.

Declassify: On input of $(\mathsf{sid}, \mathsf{Declassify}, \mathsf{varid}_y, \mathsf{varid}_x)$ from all parties:

1. If $\mathsf{type}(\mathsf{varid}_x) \neq (\mathcal{R}, s)$ or $\mathsf{type}(\mathsf{varid}_y) \neq (\mathcal{R}, -)$ then abort.
2. Assign the contents x of varid_x to varid_y and send x to the adversary if $\mathbb{C}$ contains a corrupt party.

$\mathsf{Trans}^{s \to t}$: On input of $(\mathsf{sid}, \mathsf{Trans}^{s \to t}, \mathsf{varid}_y, \mathsf{varid}_x)$ from all parties:

1. If $\mathsf{type}(\mathsf{varid}_x) \neq (\mathcal{R}, s)$ or $\mathsf{type}(\mathsf{varid}_y) \neq (\mathcal{R}, t)$ then abort.
2. Assign the contents x of varid_x to varid_y.

$\mathsf{Trans}^{t \to s}$: On input of $(\mathsf{sid}, \mathsf{Trans}^{t \to s}, \mathsf{varid}_y, \mathsf{varid}_x)$ from all parties:

1. If $\mathsf{type}(\mathsf{varid}_x) \neq (\mathcal{R}, t)$ or $\mathsf{type}(\mathsf{varid}_y) \neq (\mathcal{R}, s)$ then abort.
2. Assign the contents x of varid_x to varid_y.

Figure 3: The ideal functionality for executing a program represented by a F-Circuit

5 The MPC-FHE Protocol

Here, we combine all the components together into one protocol which executes the F-Circuit functionality.

The protocol is given in Figure 4–Figure 7, where we assume the input F-Circuit is validly typed. The protocol is given in the $\{\mathcal{F}_{\mathsf{KeyGenDec}}\}$-hybrid model, and works in the fully robust/malicious setting; i.e. the protocol should not abort. All players are assumed to be connected by authentic channels. However, if the output routine with $\mathsf{variant} = \mathsf{Enc}$ is used then the servers $\mathbb{C}$ need to be connected to the parties in $\mathbb{O}$ via **private** channels.

$\Pi_{F-Circuit}$ **(Part I)**

$\mathsf{Init}(\mathsf{sid})$:

1. The players call Init on $\mathcal{F}_{\mathsf{KeyGenDec}}$ so that all players obtain $\mathfrak{pk}$.
2. All players $\mathcal{I}_i \in \mathbb{I}$ generate a random symmetric key $\mathbf{k}_{\mathsf{i},i} \in \mathcal{R}^{\rho}$, encrypt it using $\mathsf{Enc}(\mathbf{k}_{\mathsf{i},i}, \mathfrak{pk}; r_{\mathsf{i},i})$ to obtain $\mathfrak{ct}_{\mathsf{i},i} \in \mathcal{E}^{\rho}$. The ciphertext $\mathfrak{ct}_{\mathsf{i},i}$ is *broadcast* to all players in $\mathbb{C}$.
3. All players $\mathcal{O}_i \in \mathbb{O}$ generate a random symmetric key $\mathbf{k}_{\mathsf{o},i} \in \mathcal{R}^{\rho}$, encrypt it using $\mathsf{Enc}(\mathbf{k}_{\mathsf{o},i}, \mathfrak{pk}; r_{\mathsf{o},i})$ to obtain $\mathfrak{ct}_{\mathsf{o},i} \in \mathcal{E}^{\rho}$. The ciphertext $\mathfrak{ct}_{\mathsf{o},i}$ is *broadcast* to all players in $\mathbb{C}$.
4. All players $\mathcal{C}_i \in \mathbb{C}$ generate a random symmetric key $\mathbf{k}_{\mathsf{c},i} \in \mathcal{R}^{\rho}$, encrypt it using $\mathsf{Enc}(\mathbf{k}_{\mathsf{c},i}, \mathfrak{pk}; r_{\mathsf{c},i})$ to obtain $\mathfrak{ct}_{\mathsf{c},i} \in \mathcal{E}^{\rho}$. The ciphertext $\mathfrak{ct}_{\mathsf{c},i}$ is *broadcast* to all players in $\mathbb{C}$.
5. The players in $\mathbb{I}$, $\mathbb{O}$ and $\mathbb{C}$ call the prover $\pi \leftarrow \mathsf{Prv}(\mathfrak{ct}(m, r))$ with input the respective ciphertexts ($\mathfrak{ct}_{\mathsf{i},i}$, $\mathfrak{ct}_{\mathsf{o},i}$ and $\mathfrak{ct}_{\mathsf{c},i}$), the respective plaintexts ($\mathbf{k}_{\mathsf{i},i}$, $\mathbf{k}_{\mathsf{o},i}$ and $\mathbf{k}_{\mathsf{c},i}$), and the respective randomness ($r_{\mathsf{i},i}$, $r_{\mathsf{o},i}$ and $r_{\mathsf{c},i}$), in order to obtain proofs $\pi_{\mathsf{i},i}$, $\pi_{\mathsf{o},i}$ and $\pi_{\mathsf{c},i}$.
6. The proofs are broadcast to the players in $\mathbb{C}$. The players in $\mathbb{C}$ verify the proofs using $\mathsf{Ver}(\mathfrak{ct}, \pi)$. If any proof fails then the parties in $\mathbb{C}$ replace the associated ciphertext with a default encryption of zero.
7. Set $\mathfrak{ct}_0 \leftarrow \mathfrak{ct}_{\mathsf{c},1} + \cdots + \mathfrak{ct}_{\mathsf{c},C}$, this is an encryption of $\mathbf{k}_0 = \mathbf{k}_{\mathsf{c},1} + \cdots + \mathbf{k}_{\mathsf{c},C}$.
8. Set $\mathsf{cnt}_{\mathsf{i},i} \leftarrow 0$ for all $\mathcal{I}_i \in \mathbb{I}$.
9. Set $\mathsf{cnt}_{\mathsf{o},i} \leftarrow 0$ for all $\mathcal{O}_i \in \mathbb{O}$.
10. Set $\mathsf{cnt}_{\mathsf{c}} \leftarrow 0$.

Figure 4: The protocol for executing a program represented by a F-Circuit – Part I

The connections between the players in $\mathbb{I}$ and the players in $\mathbb{C}$ are assumed to be by *reliable broadcast channels*, which are reliable in the presence of byzantine faults, including by the sender. This can be ensured by the players in $\mathbb{C}$ executing an echo-broadcast upon recieving data from a player in $\mathbb{I}$, to ensure the value has indeed been broadcast correctly. As $\mathbb{C}$ contains an honest majority, the players in $\mathbb{C}$ can take a majority vote on what the precise input is, or declare the parties input to be invalid.

For the Input and Output commands we provide two different implementations when $\mathsf{type}^2(\mathsf{varid}) = s$. We refer to the type of input/output as the $\mathsf{variant}$, which is an element of $\{\mathsf{Enc}, \mathsf{Sym}, -\}$. When the input/output value is in the clear then we assume that $\mathsf{variant} = -$.

- The first, indexed by Enc, utilizes, for input, an FHE ciphertext plus a zero knowledge proof (it therefore requires rather a large amount of data transfer), whilst for output requires the outputting party to be connected by private channels to the parties in $\mathbb{C}$ (which execute the distributed decryption functionality). Realising so many private channels may be problematic in some situations.
- The second variant, indexed by Sym, utilizes for both input and output the transciphering methodology into a ciphertext encrypted by a key which only the input (resp. output) party knows. It requires no such private channels, and enables minimal bandwidth consumption.

We see that for private output to parties we have a trade-off; either we require private channels between the output party and the parties in $\mathbb{C}$, or we only require authenticated

$\Pi_{F-Circuit}$ **(Part II)**

$\mathsf{Input}(\mathsf{sid}, \mathcal{I}_i, \mathsf{varid}, x; \mathsf{variant})$: When player $\mathcal{I}_i$ wishes to input a value x of valid type $\mathsf{type}(\mathsf{varid})$ we execute:

1. If $\mathsf{type}^2(\mathsf{varid}) = -$ then player $\mathcal{P}_i$ *broadcasts* x, in the clear, to all players in $\mathbb{C}$.
2. If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{variant} = \mathsf{Enc}$ then
   (a) Player $\mathcal{P}_i$ encrypts x using $\mathfrak{ct}_x \leftarrow \mathsf{Enc}(x, \mathfrak{pk}; r_x)$, with randomness r_x.
   (b) Player $\mathcal{P}_i$ *broadcasts* $\mathfrak{ct}_x$ to all players in $\mathbb{C}$.
   (c) Player $\mathcal{P}_i$ invokes the prover $\pi_x \leftarrow \mathsf{Prv}(\mathfrak{ct}_x, (x, r_x))$ with input the ciphertext $\mathfrak{ct}_x$, the message x and the associated randomness r_x, and broadcasts the resulting proof π_x to all players in $\mathbb{C}$.
   (d) If the proof verifies when calling $\mathsf{Ver}(\mathfrak{ct}_x, \pi_x)$, then the players in $\mathbb{C}$ store $\mathfrak{ct}_x$ in register varid, otherwise they store a default encryption of zero in register varid.
3. If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{variant} = \mathsf{Sym}$ then
   (a) Player $\mathcal{P}_i$ encrypts x using

$$\mathbf{c}_x \leftarrow \mathsf{Sym}(\ x, \mathbf{k}_{\mathsf{i},i};\ \mathsf{sid}, \mathsf{cnt}_{\mathsf{i},i}\ ).$$

   (b) Player $\mathcal{P}_i$ *broadcasts* $\mathbf{c}_x$ to all players in $\mathbb{C}$.
   (c) The players in $\mathbb{C}$ transcipher $\mathbf{c}_x$ from an encryption under Sym to an encryption under Enc by computing

$$\mathfrak{ct}_x \leftarrow \mathsf{Sym}^{-1}(\ \mathbf{c}_x, \mathfrak{ct}_{\mathsf{i},i};\ \mathsf{sid}, \mathsf{cnt}_{\mathsf{i},i}\ ).$$

   (d) Incremement $\mathsf{cnt}_{\mathsf{i},i}$.

   Note, if any of the proofs do not verify, or the broadcasts are invalid then the players in $\mathbb{C}$ store a deterministic default value, say zero (resp. a homomorphic encryption of zero) in the register varid.

Figure 5: The protocol for executing a program represented by a F-Circuit – Part II

channels, but we require the computing parties to execute a transciphering operation. Both options require a distributed decryption operation; one with private output and one with public output.

$\Pi_{F-Circuit}$ **(Part III)**

$\mathsf{Output}(\mathsf{sid}, \mathcal{O}_i, \mathsf{varid}; \mathsf{variant})$: When player $\mathcal{O}_i$ is expecting an output value x of valid type $\mathsf{type}(\mathsf{varid})$ we execute:

1. If $\mathsf{type}^2(\mathsf{varid}) = -$ then
   (a) The contents of register varid is sent to player $\mathcal{O}_i$ by all players in $\mathbb{C}$.
   (b) Player $\mathcal{O}_i$ takes the majority verdict as the value of the register sent.
   (c) The value sent is the output value.
2. If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{varid} = \mathsf{Enc}$ then
   (a) All players in $\mathbb{C} \cup \{\mathcal{O}_i\}$ invoke $\mathsf{DistDecrypt}(\mathsf{sid}, \mathfrak{ct}_x, \{\mathcal{O}_i\})$ on $\mathcal{F}_{\mathsf{KeyGenDec}}$ where $\mathfrak{ct}_x$ is the contents of register varid.
   (b) The party $\mathcal{O}_i$ receives the output plaintext value, and takes this as the output value. *Note, that $\mathcal{O}_i$ only receives the value implies* **private channels** *between the entities in $\mathbb{C}$ and the entities in $\mathbb{O}$.*
3. If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{varid} = \mathsf{Sym}$ then
   (a) All players in $\mathbb{C}$ take the contents $\mathfrak{ct}_x$ of register varid and computes homomorphically
   $$\mathfrak{ct}'_x \leftarrow \mathsf{Sym}(\ \mathfrak{ct}_x, \mathfrak{ct}_{\mathfrak{o},i};\ \mathsf{sid}, \mathsf{cnt}_{\mathfrak{o},i}\ ).$$
   (b) All players in $\mathbb{C}$ invoke $\mathbf{c} \leftarrow \mathsf{DistDecrypt}(\mathsf{sid}, \mathfrak{ct}'_x, \mathbb{C})$ on $\mathcal{F}_{\mathsf{KeyGenDec}}$.
   (c) The value $\mathbf{c}$ is sent to player $\mathcal{O}_i$ by the players in $\mathbb{C}$, with $\mathcal{O}_i$ taking the majority verdict as the value of the ciphertext sent.
   (d) Player $\mathcal{O}_i$ decrypts $\mathbf{c}$ to obtain x by computing
   $$x \leftarrow \mathsf{Sym}^{-1}(\ \mathbf{c}, \mathbf{k}_{\mathfrak{o},i};\ \mathsf{sid}, \mathsf{cnt}_{\mathfrak{o},i}\ ).$$
   (e) Incremement $\mathsf{cnt}_{\mathfrak{o},i}$.

Figure 6: The protocol for executing a program represented by a F-Circuit – Part III

$\Pi_{F-Circuit}$ (Part IV)

$\mathsf{Add}(\mathsf{sid}, \mathsf{varid}_z, \mathsf{varid}_x, \mathsf{varid}_y)$:

1. The players in $\mathbb{C}$ retrieve x from varid_x and y from varid_y and store $z = x + y$ in varid_z.

$\mathsf{Mult}(\mathsf{sid}, \mathsf{varid}_z, \mathsf{varid}_x, \mathsf{varid}_y)$:

1. The players in $\mathbb{C}$ retrieve x from varid_x and y from varid_y and store $z = x \cdot y$ in varid_z.

$\mathsf{Declassify}(\mathsf{sid}, \mathsf{varid}_y, \mathsf{varid}_x)$:

1. The players in $\mathbb{C}$ retreive $\mathfrak{ct}_x$ from varid_x and execute $\mathsf{DistDecrypt}(\mathsf{sid}, \mathfrak{ct}_x, \mathbb{C})$ on $\mathcal{F}_{\mathsf{KeyGenDec}}$.
2. The output y they receive is assigned to the register varid_y.

$\mathsf{Trans}^{s \to t}(\mathsf{sid}, \mathsf{varid}_y, \mathsf{varid}_x)$:

1. The players in $\mathbb{C}$ retrieve $\mathfrak{ct}_x$ from varid_x, recall this is an FHE-encrypted ciphertext.
2. The players in $\mathbb{C}$ homomorphically compute
$$\mathfrak{ct}'_x \leftarrow \mathsf{Sym}(\ \mathfrak{ct}_x, \mathfrak{ct}_0;\ \mathsf{sid}, \mathsf{cnt}_{\mathfrak{c}}\ ).$$
3. The players in $\mathbb{C}$ execute $\mathbf{c} \leftarrow \mathsf{DistDecrypt}(\mathsf{sid}, \mathfrak{ct}'_x, \mathbb{C})$ on $\mathcal{F}_{\mathsf{KeyGenDec}}$.
4. The output $\mathbf{c}$ they receive (along with the counter $\mathsf{cnt}_{\mathfrak{c}}$) is assigned to the register varid_y.
5. Incremement $\mathsf{cnt}_{\mathfrak{c}}$.

$\mathsf{Trans}^{t \to s}(\mathsf{sid}, \mathsf{varid}_y, \mathsf{varid}_x)$:

1. The players in $\mathbb{C}$ retrieve $\mathsf{cnt}'_{\mathfrak{c}} \| \mathbf{c}_x$ from varid_x, recall this is a Sym-encrypted ciphertext with index $\mathsf{cnt}'_{\mathfrak{c}}$.
2. The players in $\mathbb{C}$ homomorphically compute
$$\mathfrak{ct}_y \leftarrow \mathsf{Sym}^{-1}(\ \mathbf{c}_x, \mathfrak{ct}_0;\ \mathsf{sid}, \mathsf{cnt}'_{\mathfrak{c}}\ ).$$
3. The output $\mathfrak{ct}_y$ is assigned to the register varid_y.

Figure 7: The protocol for executing a program represented by a F-Circuit – Part IV

6 Security Proof

The security proof for our protocol will follow the rough outline of the proof of the *offline phase* for the SPDZ protocol from [DPSZ12]. The formal statement we aim to prove is

Theorem 1. *In the random oracle model*[14][15]*, the protocol $\Pi_{F-Circuit}$ (in Figure 4–Figure 7) UC realizes the functionality $\mathcal{F}_{F-Circuit}$ from Figure 3 in the $\{ \mathcal{F}_{\mathsf{KeyGenDec}} \}$-hybrid model, assuming the underlying FHE encryption scheme is IND-KEY secure, and the cipher* Sym *is IND-CPA.*

This proof is in the UC model, we so we first outline the basic overview. As well as the honest parties executing a protocol we have an adversarial party $\mathcal{A}$ which performs the actions for the parties which are adversarially controlled. The adversary $\mathcal{A}$ mounts an attack on the protocol, by statically corrupting as many input and output players as it wants, as well as up to t of the computing parties[16].

In addition there is an additional entity $\mathcal{Z}$ called the environment. The environment sits in one of two worlds; a real world or an ideal world. The goal of the enviroment is to output a single bit. If $\mathcal{Z}$ outputs one, then it thinks it is sitting in a real world. On the other hand if $\mathcal{Z}$ outputs zero, then it thinks it is sitting in a simulated world.

When $\mathcal{Z}$ is sitting in the *real world* it selects the honest parties inputs, which it gives to the honest parties. The environment also receives the outputs of the honest parties at the end of the protocol execution. However, the environment does not get to see the random tapes/internal variables (e.g. the keys, key shares etc) of the honest parties. The environment executes the adversarial entity and during the attack, $\mathcal{Z}$ takes complete control over the actions of the corrupted parties.

When $\mathcal{Z}$ is sitting in the *ideal world*, there is now another entity called a simulator $\mathcal{S}$. The simulator must simulate the environment's view of the protocol so that it looks like what $\mathcal{Z}$ would see in a real attack. The simulator has access to the ideal functionality $\mathcal{F}_{F-Circuit}$ from Figure 3. The environment in the ideal world still selects the honest parties inputs, which it gives to the (dummy) honest parties. If the inputs are private, these inputs are passed directly to the ideal functionality, and are not available to the simulator. Again during the attack, the environment takes complete control over the actions of the corrupted parties. Once again, at the end of the protocol, the environment $\mathcal{Z}$ gets access to the outputs of the honest parties, by having them directly passed to $\mathcal{Z}$ from the ideal functionality. The simulator $\mathcal{S}$ does not get access to the private outputs of the honest parties, but does get access to the public outputs of all parties.

Our protocol $\Pi_{F-Circuit}$, from Figure 4–Figure 7, is said to (computationally) securely implement $\mathcal{F}_{F-Circuit}$, if a poly-time bounded environment $\mathcal{Z}$ outputs one with essentially the same probability in the real as the ideal process.

In a $\mathcal{G}$-hybrid model, the protocol executes just as in the real world, however the parties have access to an ideal functionality $\mathcal{G}$ for some sub-task. During the protocol, the parties communicate with $\mathcal{G}$ as in the ideal world. The main UC-composition theorem [Can00, Can01] says that an ideal functionality proved secure in the $\mathcal{G}$-hybrid model, will be secure when the sub-task represented by $\mathcal{G}$ is realized by a UC-secure protocol that realizes $\mathcal{G}$.

[14]We model the hash functions used in the ZKPoKs as random oracles, as discussed earlier.

[15]If using vector commitment based ZKPoKs then one also needs to add the Algebraic Group Model as well here.

[16]Static corruption implies that *all* the parties are actually defined at the start of the protocol. This can be relaxed, by for example allowing the adversary to add in new parties to $\mathbb{I}$ and $\mathbb{O}$ dynamically (but not new parties into $\mathbb{C}$), which are either honest or corrupt.

$\mathcal{S}_{F-Circuit}$ **(Part I)**

$\mathsf{Init}(\mathsf{sid})$:

1. The simulator simulates the call to $\mathcal{F}_{\mathsf{KeyGenDec}}$ by internally running the algorithm KeyGen, in order to obtain a key pair $(\mathfrak{pk}, \mathfrak{sk})$, as well as shares $\langle \mathfrak{sk} \rangle_i$. The $\mathfrak{pk}$ is sent to all players, and the $\langle \mathfrak{sk} \rangle_i$ for corrupt $\mathcal{C}_i$ are sent to the adversary. The global secret key $\mathfrak{sk}$ is stored by the simulator.
2. The simulator executes all the steps of Init of $\Pi_{F-Circuit}$ for the honest players.
3. The simulator on receiving $\mathfrak{ct}_{\mathsf{i},i}$, $\mathfrak{ct}_{\mathsf{o},i}$ or $\mathfrak{ct}_{\mathsf{c},i}$ from adversarial players, along with their zero-knowledge proofs, $\pi_{\mathsf{i},i}$, $\pi_{\mathsf{o},i}$ or $\pi_{\mathsf{c},i}$, it
   (a) Checks the proofs are correct by calling $\mathsf{Ver}(\mathfrak{ct}_{\mathsf{i},i}, \pi_{\mathsf{i},i})$. If any verification fails then the associated ciphertext is replaced with a default encryption of zero.
   (b) Decrypt the ciphertext using $\mathfrak{sk}$, so as to obtain $\mathbf{k}_{\mathsf{i},i}$, $\mathbf{k}_{\mathsf{o},i}$, and $\mathbf{k}_{\mathsf{c},i}$
4. This means that the simulator knows $\mathbf{k}_{\mathsf{i},i}$, $\mathbf{k}_{\mathsf{o},i}$, and $\mathbf{k}_{\mathsf{c},i}$ for all i, and hence also knows $\mathbf{k}_0 = \mathbf{k}_{\mathsf{c},1} + \cdots + \mathbf{k}_{\mathsf{c},C}$.

Figure 8: The simulator for $\mathcal{F}_{F-Circuit}$ – Part I

$\mathcal{S}_{F-Circuit}$ **(Part II)**

$\mathsf{Input}(\mathsf{sid}, \mathcal{I}_i, \mathsf{varid}, x; \mathsf{variant})$:

1. If $\mathcal{I}_i$ is honest:
   (a) If $\mathsf{type}^2(\mathsf{varid}) = -$ then the functionality will output x to the adversary, thus the simulator will also learn x. The simulator essentially does nothing for such operations.
   (b) If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{variant} = \mathsf{Enc}$ then the simulator, who does not know x, generates an encryption $\mathfrak{ct}_x$ of a random value x, using randomness r_x, along with a valid proof obtained by executing $\pi_x \leftarrow \mathsf{Prv}(\mathfrak{ct}_x, (x, r_x))$. The simulator passes $\mathfrak{ct}_x$ and π_x to any adversarial parties in $\mathbb{C}$.
   (c) If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{variant} = \mathsf{Sym}$ then the simulator, who again does not know x, but does know $\mathbf{k}_{\mathsf{i},i}$ generates an encryption $\mathbf{c}_x$ of a random value under the encryption scheme Sym, and passes this to any adversarial parties in $\mathbb{C}$.
   (d) The parties in $\mathbb{C}$ then process the incoming value as specified in the protocol.
2. If $\mathcal{I}_i$ is dishonest:
   (a) What ever value of $\mathsf{type}(\mathsf{varid})$ is sent, the simulator can extract the underlying value x, either because it is in clear, or because $\mathcal{S}$ knows $\mathfrak{sk}$, or because $\mathcal{S}$ knows $\mathbf{k}_{\mathsf{i},i}$.
   With the substitution for zero in the case of the adversary not sending a valid proof, or not broadcasting a consistent value etc.
   (b) The value x is then passed to $\mathcal{F}_{F-Circuit}$ by calling $(\mathsf{sid}, \mathsf{Input}, \mathcal{I}_i, \mathsf{varid}, x)$.

Figure 9: The simulator for $\mathcal{F}_{F-Circuit}$ – Part II

$\mathcal{S}_{F-Circuit}$ **(Part III)**

```
Output(sid, $\mathcal{O}_i$, varid; variant):
    1. If $\mathcal{O}_i$ is honest:
        (a) If $\mathsf{type}^2(\mathsf{varid}) = -$ then the simulator knows the value $x$, and can
            thus transmit the 'in clear' communication to $\mathcal{I}_i$ for honest parties
            in $\mathbb{C}$.
        (b) If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{variant} = \mathsf{Enc}$ then the simulator simulates
            the operation by utilizes the distributed decryption functionality.
        (c) If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{variant} = \mathsf{Sym}$ then the simulator simulates
            a call to DistDecrypt with output being a public random ciphertext
            $\mathbf{c}$ under Sym for the key $\mathbf{k}_{\mathsf{o},i}$.
    2. If $\mathcal{O}_i$ is dishonest:
        (a) If $\mathsf{type}^2(\mathsf{varid}) = -$ then the simulator knows the value $x$, and can
            thus simulate the transmission of the 'in clear' communication to $\mathcal{I}_i$
            from the honest parties in $\mathbb{C}$.
        (b) If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{variant} = \mathsf{Enc}$ then the simulator simu-
            lates a call to DistDecrypt, for the output value obtained by calling
            $(\mathsf{sid}, \mathsf{Output}, \mathcal{O}_i, \mathsf{varid})$ on $\mathcal{F}_{F-Circuit}$.
        (c) If $\mathsf{type}^2(\mathsf{varid}) = s$ and $\mathsf{variant} = \mathsf{Sym}$ then the simulator obtains $x$
            by calling $(\mathsf{sid}, \mathsf{Output}, \mathcal{O}_i, \mathsf{varid})$ on $\mathcal{F}_{F-Circuit}$. The value $x$ is then
            encrypting using Sym with the key $\mathbf{k}_{o,i}$ to obtain $\mathbf{c}$. Finally, the
            simulator simulates a call to DistDecrypt for the output value $\mathbf{c}$.
```

Figure 10: The simulator for $\mathcal{F}_{F-Circuit}$ – Part III

$\mathcal{S}_{F-Circuit}$ **(Part IV)**

```
Add(sid, varid$_z$, varid$_x$, varid$_y$):
    1. There is nothing to simulate here
Mult(sid, varid$_z$, varid$_x$, varid$_y$):
    1. There is nothing to simulate here, the honest players just follow the
       protocol.
Declassify(sid, varid$_y$, varid$_x$):
    1. The simulator simulates a call to DistDecrypt, for the output value
       obtained by calling $(\mathsf{sid}, \mathsf{Declassify}, \mathsf{varid}_y, \mathsf{varid}_x)$ on $\mathcal{F}_{F-Circuit}$.
Trans$^{s \to t}$(sid, varid$_y$, varid$_x$):
    1. Generate the ciphertext $\mathfrak{ct}'_x$ as in the real protocol.
    2. Simulate a call to DistDecrypt, for the output value $\mathbf{c}$ a random ciphertext
       output by Sym under the key $\mathbf{k}_0$.
Trans$^{t \to s}$(sid, varid$_y$, varid$_x$):
    1. There is nothing to simulate here, the honest players just follow the
       protocol.
```

Figure 11: The simulator for $\mathcal{F}_{F-Circuit}$ – Part IV

6.1 The Simulator $\mathcal{S}$

The simulator is given in Figure 8–Figure 11. It will play the honest player's parts, whilst the environment $\mathcal{Z}$ will play the parts of the corrupt players[17]. The simulator runs an internal copy of $\mathcal{F}_{\mathsf{KeyGen}}$ and $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{enc}}}$, in order to simulate calls to these functionalities.

The key problem in creating the simulator, is that to call the ideal functionality, the simulator needs to somehow extract the dishonest parties private inputs from the output of the adversary, in order to pass the adversarial inputs to the ideal functionality. However, since the simulator knows the underlying secret key of the FHE scheme, via the functionality $\mathcal{F}_{\mathsf{KeyGen}}$, it can decrypt every FHE ciphertext sent on behalf of the corrupt players. This can then be used to obtain the required input, for the corrupt parties, to the functionality $\mathcal{F}_{F-Circuit}$.

6.1.1 Discussion:

There is something a bit intuitively weird going on with the simulation here, for those not used to UC proofs, which we now elaborate on. Consider the program executed between two input/output parties $\mathcal{P}_1$ and $\mathcal{P}_2$, of which $\mathcal{P}_1$ is honest and $\mathcal{P}_2$ is dishonest:

```
Type (R,s): varid_x, varid_y, varid_z;
Type (R,-): varid_d;

Input(P1,varid_x,x; Sym);        <- Line 1
Input(P2,varid_y,y; Sym);        <- Line 2
varid_z = varid_x + varid_y;     <- Line 3
varid_d = Declassify(varid_z);  <- Line 4
```

In Line 1 of our sample F-Circuit the honest party enters a ciphertext $\mathbf{c}_x$ which encrypts their input x under the cipher Sym, and then this is transformed into a ciphertext $\mathfrak{ct}_x$ encrypting their input x under Enc by computing

$$\mathfrak{ct}_x \leftarrow \mathsf{Sym}(\ \mathbf{c}_x, \mathfrak{ct}_{\mathsf{i},1};\ \mathsf{sid}, \mathsf{cnt}_{\mathsf{i},1}\ ).$$

Then in Line 4 all parties obtain the value of z. But party $\mathcal{P}_2$ knows y and hence from z can compute x. Thus it seems the above program seems to provide a decryption oracle, both for the ciphertext $\mathbf{c}_x$ encrypting x under Sym, and the ciphertext $\mathfrak{ct}_x$ encrypting x under Enc.

Thus it seems that our Declassify algorithm provides a decryption oracle for the adversary for the scheme Sym *and* the scheme Enc. But neither of these schemes are IND-CCA, they are only IND-CPA. This can at first sight seem very weird indeed; that an IND-CPA scheme is enough to protect against what appears to be some form of decryption oracle. But it can be explained by two facts: The first fact is that in the security proof we never need to call a decryption oracle to decrypt any ciphertext encrypted under Sym or Enc for an unknown secret key. The second fact is that the Declassify algorithm calls are actually answered in the proof by calling the ideal functionality. Thus the information which is revealed by the decryption oracle, and any information which can be derived from such queries, are allowed by the security model of the ideal functionality.

For example in a normal CCA attack against an additive homomorphic encryption scheme, one might encrypt m_0 or m_1, and then ask the decryption oracle to decrypt the ciphertext which decrypts to $2 \cdot m_b$, by using the additive homomorphic property. But this is perfectly acceptable in our model, as if the ideal functionality allows the leaking of $2 \cdot m_b$, then it also allows the adversary to learn m_b.

[17]Thus for simplicity of exposition we assume only one adversarial entity, and do not split into an "environment" and an "adversary".

There is then some more (at first sight) weirdness in the simulation. Again look at Line 1 of our example F-Circuit, as $\mathcal{P}_1$ is honest the simulator creates a random ciphertext $\mathbf{c}_x$ which purports to encrypt x; but it clearly does not. It is actually an encryption of a value x'. During the following processing of Line 1, the parties in $\mathbb{C}$ will produce an encryption $\mathfrak{ct}_x$ by computing

$$\mathfrak{ct}_x \leftarrow \mathsf{Sym}^{-1}(\ \mathbf{c}_x, \mathfrak{ct}_{\mathsf{i},1};\ \mathsf{sid}, \mathsf{cnt}_{\mathsf{i},1}\ ),$$

but this is neither an encryption of x or x', Then in Line 4 the value x is revealed to the adversary.

It would appear, at first sight, that this *could* be used by the adversary to tell that the values $\mathbf{c}_x$ and $\mathfrak{ct}_x$ are fake. However, security of $\mathbf{c}_x$ is maintained since we assume, in the proof, that the function Sym cannot be distinguished from an ideal random function. The security of $\mathfrak{ct}_x$ is maintained by a similar IND-CPA style argument (actually our IND-KEY definition from earlier) for the homomorphic encryption scheme.

6.2 The Proof

We now present a proof sketch of Theorem 1. As pointed out earlier our proof follows roughly the same idea as the proof sketch of the offline phase in [DPSZ12]. We only present a sketch, leaving out the specifics of accounting for the acummulation of different extraction and accumulation errors, in order to make the main ideas clearer for the reader.

We first assume that there is an environment $\mathcal{Z}$ that can distinguish between the real process and the simulated/ideal process. In other words we have, for some $\mathcal{Z}$, that

$$\mathsf{Adv}(\mathcal{Z}) := \Pr[\ 1 \leftarrow \mathcal{Z}(\ \text{Real}\ )\ ]\ -\ \Pr[\ 1 \leftarrow \mathcal{Z}(\ \text{Ideal}\ )\ ] \geq \epsilon.$$

We will use $\mathcal{Z}$ to construct an algorithm B which breaks in the IND-KEY security of the FHE scheme. By contradiction we can then conclude that our initial assumption of $\mathcal{Z}$ existing is false.

Algorithm B takes as input a public key $\mathfrak{pk}^*$ and is asked to determine whether $\mathfrak{pk}^*$ comes from a valid execution of $\mathsf{KeyGen}(1^\kappa)$ or the execution of $\mathsf{KeyGen}^*(1^\kappa)$, which recall is an algorithm which produces a "meaningless" public key, i.e. one for which the encryption algorithm produces ciphertexts which contain no information on the underlying message. Thus, B's goal is to determine whether $\mathfrak{pk}^* = \mathfrak{pk}$, for the an output $\mathfrak{pk}$ of $\mathsf{KeyGen}(1^\kappa)$, or $\mathfrak{pk}^* = \widetilde{\mathfrak{pk}}$, for an output $\widetilde{\mathfrak{pk}}$ of $\mathsf{KeyGen}^*(1^\kappa)$. Recall the public keys output by these two algorithms are themselves indistinguishable.

The algorithm B will execute $\mathcal{Z}$, in that algorithm B will play for the honest parties, whilst $\mathcal{Z}$ will play for the dishonest parties. The main thing that B does is that it uses it's input public key $\mathfrak{pk}^*$, instead of calling $\mathcal{F}_{\mathsf{KeyGen}}$ in step 1 of Init in Figure 8. For the shares $\mathfrak{sk}_i$ for the dishonest players, it sends random values to the environment $\mathcal{Z}$. Thus B needs to operate without needing access to the underlying secret key $\mathfrak{sk}$.

Algorithm B then selects a bit b. When $b = 0$ it tries to emulate to $\mathcal{Z}$ an execution of the 'Ideal' environment, when $b = 1$ it tries to emulate an execution of the 'Real' environment. Algorithm B will output one (i.e. it thinks the input key $\mathfrak{pk}^*$ is a real one, i.e. $\mathfrak{pk}$), if when it selects $b = 0$ algorithm $\mathcal{Z}$ outputs zero (i.e. $\mathcal{Z}$ thinks it is in the ideal world), and when it selects $b = 1$ algorithm $\mathcal{Z}$ outputs one (i.e. $\mathcal{Z}$ thinks it is in the real world).

$\underline{b = 0:}$ Algorithm B in this case runs a copy of the simulator, and it does exactly what the simulator would do. Except it is unable to decrypt the ciphertexts encrypted under $\mathfrak{pk}^*$, as B does not have the associated private key.

So in line 3 of Init and line 2a of Input, algorithm B uses the knowledge extractor for the associated zero-knowledge proofs to extract the underlying message (with negligible probability of failure). In particular[18] it rewinds the prover in the zero-knowledge proof (i.e. $\mathcal{Z}$). The algorithm B is allowed to do this, since it is running its own copy of $\mathcal{Z}$. It can respond to the challenges in different ways, in order to extract the value, since B controls the random oracle used to generate the proof challenges.

The zero-knowledge proofs that honest players execute in the protocol are simulated by B, using the algorithm Sim, and not executed for real (even on the random inputs chosen by the simulator).

It is clear that from $\mathcal{Z}$'s perspective in the case when $b = 0$ and the public key is valid, i.e. $\mathfrak{pk}^* = \mathfrak{pk}$, then $\mathcal{Z}$ is running in the simulated execution. Thus,

$$\Pr[\ 1 \leftarrow \mathcal{B}(\mathfrak{pk}) \mid b = 0\ ] = \Pr[\ 0 \leftarrow \mathcal{Z}(\ \text{Ideal}\ )\ ] + \delta_0,$$

where δ_0 accounts for any probability of failure in extracting the underlying messages for the dishonest parties or errors in the zero-knowledge proof simulations, and is thus negligible.

When $\mathfrak{pk}^* = \widetilde{\mathfrak{pk}}$ the FHE encryptions contain statistically no information about the messages contained inside them. Also the zero-knowledge proofs reveal no information, as they were produced via simulations and not via actual values. In addition, as Sym is assumed to be IND-CPA, we can do a game hop to ensure that the encryptions of messages m under the cipher Sym for honest players symmetric keys, or the key $\mathbf{k}_0$, also contain no information about the underlying messages. Thus we have, since the algorithm $\mathcal{B}$ sees no information about the messages it can at best output one with probability one half,

$$\Pr[\ 1 \leftarrow \mathcal{B}(\widetilde{\mathfrak{pk}}) \mid b = 0\ ] = \frac{1}{2} + \delta_0',$$

where δ_0' is essentially as above, but also includes terms related to the IND-CPA advantage of the function Sym.

$\underline{b = 1:}$ When $b = 1$ algorithm B will run the honest players precisely as in the real algorithm; except the inputs for the honest players will be provided by the environment $\mathcal{Z}$ and any zero-knowledge proofs will be provided by running the zero-knowledge simulator, Sim, and not by running the real prover, Prv.

Algorithm B can now provide a simulation to the environment $\mathcal{Z}$, without needing recourse to the ideal functionality. It first extracts the inputs from the dishonest players inputs, just as in the case $b = 0$ above, by rewinding. Given the honest and dishonest players inputs the algorithm $\mathcal{B}$ can simulate the $\mathsf{Declassify}$ queries by running the F-Circuit in the "clear".

It is then clear that from $\mathcal{Z}$'s perspective in the case when $b = 1$ and the public key is valid, i.e. $\mathfrak{pk}^* = \mathfrak{pk}$, then $\mathcal{Z}$ is running in the real execution. Thus,

$$\Pr[\ 1 \leftarrow \mathcal{B}(\mathfrak{pk}) \mid b = 1\ ] = \Pr[\ 1 \leftarrow \mathcal{Z}(\ \text{Real}\ )\ ] + \delta_1,$$

where δ_1 accounts for any probability of failure in extracting the underlying messages for the dishonest parties or errors in the zero-knowledge proof simulations, and is thus negligible.

Again, when $\mathfrak{pk}^* = \widetilde{\mathfrak{pk}}$ the FHE encryptions contain statistically no information about the messages contained inside them, neither do the zero-knowledge proofs (as they are simulations), and neither do the encryptions under Sym (after a suitable game hop). Thus we have, for some small negligible constant δ_1',

$$\Pr[\ 1 \leftarrow \mathcal{B}(\widetilde{\mathfrak{pk}}) \mid b = 1\ ] = \frac{1}{2} + \delta_1'.$$

[18]For ZKPoKs based on Fiat-Shamir transforms, for those based on vector commitments one extracts the witness by inspecting the lists maintained in the AGM.

Combining these cases together we have, assuming $\Pr[b = 1] = \Pr[b = 0] = 1/2$,

$$\begin{aligned}
2 \cdot \mathsf{Adv}(\mathsf{B}) &= 2 \cdot \Big(\Pr[\ 1 \leftarrow \mathcal{B}(\mathfrak{pk})\ ] - \Pr[\ 1 \leftarrow \mathcal{B}(\widetilde{\mathfrak{pk}})\ ]\ \Big), \\
&= \Big(\Pr[\ 1 \leftarrow \mathcal{B}(\mathfrak{pk}) \mid b = 0\ ] \\
&\qquad + \Pr[\ 1 \leftarrow \mathcal{B}(\mathfrak{pk}) \mid b = 1\ ]\Big) \\
&\quad - \Big(\Pr[\ 1 \leftarrow \mathcal{B}(\widetilde{\mathfrak{pk}}) \mid b = 0\ ] \\
&\qquad + \Pr[\ 1 \leftarrow \mathcal{B}(\widetilde{\mathfrak{pk}}) \mid b = 1\ ]\Big), \\
&= \Big(\Pr[\ 0 \leftarrow \mathcal{Z}(\ \mathrm{Ideal}\ )\ ] + \delta_0 \Big) \\
&\quad + \Big(\Pr[\ 1 \leftarrow \mathcal{Z}(\ \mathrm{Real}\ )\ ] + \delta_1 \Big) \\
&\quad - \Big(\frac{1}{2} + \delta_1' \Big) \\
&\quad - \Big(\frac{1}{2} + \delta_0' \Big), \\
&= \Big(\Pr[\ 1 \leftarrow \mathcal{Z}(\ \mathrm{Real}\ )\ ] + \Big(1 - \Pr[\ 1 \leftarrow \mathcal{Z}(\ \mathrm{Ideal}\ )\ ] \Big) \Big) \\
&\quad + (\delta_0 + \delta_1 - \delta_0' - \delta_1') - 1, \\
&= \Big(\Pr[\ 1 \leftarrow \mathcal{Z}(\ \mathrm{Real}\ )\ ] - \Pr[\ 1 \leftarrow \mathcal{Z}(\ \mathrm{Ideal}\ )\ ] \Big), \\
&\quad + \Big(\delta_0 + \delta_1 - \delta_0' - \delta_1' \Big), \\
&= \mathsf{Adv}(\mathcal{Z}) + \delta, \\
&\geq \epsilon + \delta
\end{aligned}$$

So if ϵ is non-negligible, i.e. the simulation can be distinguished by the environment $\mathcal{Z}$, then $\mathsf{Adv}(\mathsf{B})$ is also non-negligible and the FHE scheme is not IND-KEY secure. As we assume that the scheme is IND-KEY secure, this implies that ϵ cannot be non-negligible, and thus our protocol is secure.

Acknowledgements

The author would like to thank Benoit Libert, Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy for helpful conversations during the work on this paper. The work of the author was supported by CyberSecurity Research Flanders with reference number VR20192203, and by the FWO under an Odysseus project GOH9718N.

References

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 153–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-53018-4_6

[AJL$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-29011-4_29

[AJW11] Gilad Asharov, Abhishek Jain, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. Cryptology ePrint Archive, Report 2011/613, 2011. `https://eprint.iacr.org/2011/613`.

[BCS19] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*, volume 11959 of *Lecture Notes in Computer Science*, pages 274–302, Waterloo, ON, Canada, August 12–16, 2019. Springer, Heidelberg, Germany. doi: 10.1007/978-3-030-38471-5_12

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 201–218, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-11799-2_13

[BdO$^+$21] Karim Baghery, Cyprien de Saint Guilhem, Emmanuela Orsini, Nigel P. Smart, and Titouan Tanguy. Compilation of function representations for secure computing paradigms. In Kenneth G. Paterson, editor, *Topics in Cryptology – CT-RSA 2021*, volume 12704 of *Lecture Notes in Computer Science*, pages 26–50, Virtual Event, May 17–20, 2021. Springer, Heidelberg, Germany. doi: 10.1007/978-3-030-75539-3_2

[BGG$^+$18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 565–596, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. doi: 10.1007/978-3-319-96884-1_19

[BGGK17] Dan Boneh, Rosario Gennaro, Steven Goldfeder, and Sam Kim. A lattice-based universal thresholdizer for cryptographic systems. Cryptology ePrint Archive, Report 2017/251, 2017. `https://eprint.iacr.org/2017/251`.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery. doi: 10.1145/2090236.2090262

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 645–677, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. doi: 10.1007/978-3-319-70500-2_22

[BK11] Dan Bogdanov and Liina Kamm. Constructing privacy-preserving information systems using secure multiparty computation. Cybernetica Research Report T-4-13, 2011.

[BKL+14] Dan Bogdanov, Liina Kamm, Sven Laur, Pille Pruulmann-Vengerfeldt, Riivo Talviste, and Jan Willemson. Privacy-preserving statistical data analysis on federated databases. In *Proceedings of the Annual Privacy Forum. APF'14*, volume 8450 of *LNCS*, pages 30–55. Springer, 2014.

[BLLN13] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In Martijn Stam, editor, *14th IMA International Conference on Cryptography and Coding*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64, Oxford, UK, December 17–19, 2013. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-45239-0_4

[BNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008: 15th Conference on Computer and Communications Security*, pages 257–266, Alexandria, Virginia, USA, October 27–31, 2008. ACM Press. doi: 10.1145/1455770.1455804

[BOS23] Thibault Balenbois, Jean-Baptiste Orfila, and Nigel P. Smart. Trivial transciphering with trivium and tfhe. Cryptology ePrint Archive, Paper 2023/980, 2023. `https://eprint.iacr.org/2023/980`.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 190–213, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-53018-4_8

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-32009-5_50

[BS23] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. Cryptology ePrint Archive, Report 2023/016, 2023. `https://eprint.iacr.org/2023/016`.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `https://eprint.iacr.org/2000/067`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press. doi: 10.1109/SFCS.2001.959888

[Cat21] Octavian Catrina. Complexity and performance of secure floating-point polynomial evaluation protocols. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021: 26th European Symposium on Research in Computer Security, Part II*, volume 12973 of *Lecture Notes in Computer Science*, pages 352–369, Darmstadt, Germany, October 4–8, 2021. Springer, Heidelberg, Germany. doi: 10.1007/978-3-030-88428-4_18

[CCF^{+}16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In Thomas Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 313–333, Bochum, Germany, March 20–23, 2016. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-52993-5_16

[CCK23] Jung Hee Cheon, Wonhee Cho, and Jiseung Kim. Improved universal thresholdizer from threshold fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/545, 2023. `https://eprint.iacr.org/2023/545`.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from TFHE. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 446–472, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany. doi: 10.1007/978-3-030-34621-8_16

[Cd10] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10: 7th International Conference on Security in Communication Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199, Amalfi, Italy, September 13–15, 2010. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-15317-4_13

[CDKS19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 395–412, London, UK, November 11–15, 2019. ACM Press. doi: 10.1145/3319535.3363207

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020. doi: 10.1007/s00145-019-09319-x

[CJL^{+}20] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE. In *8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography (WAHC 2020)*, pages 57–63. Leibniz Universität IT Services, 2020.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. doi: 10.1007/978-3-319-70694-8_15

[CLO^{+}13] Ashish Choudhury, Jake Loftus, Emmanuela Orsini, Arpita Patra, and Nigel P. Smart. Between a rock and a hard place: Interpolating between MPC and FHE. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 221–240, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-42045-0_12

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 630–656, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-48000-7_31

[Coh16] Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 183–207, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-49387-8_8

[CS10] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Heidelberg, Germany.

[CSS$^+$22] Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. Efficient threshold FHE with application to real-time systems. Cryptology ePrint Archive, Report 2022/1625, 2022. `https://eprint.iacr.org/2022/1625`.

[CsW19] Ran Cohen, abhi shelat, and Daniel Wichs. Adaptively secure MPC with sublinear communication complexity. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 30–60, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. doi: 10.1007/978-3-030-26951-7_2

[CZW17] Long Chen, Zhenfeng Zhang, and Xueqing Wang. Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 597–627, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. doi: 10.1007/978-3-319-70503-3_20

[DDE$^+$23] Morten Dahl, Daniel Demmler, Sarah Elkazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Samual Tap, and Michael Walter. Noah's ark: Efficient threshold-fhe using noise flooding. Cryptology ePrint Archive, Paper 2023/815, 2023. `https://eprint.iacr.org/2023/815`.

[De 06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC 2006: 9th International Conference on Information Security*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186, Samos Island, Greece, August 30 – September 2, 2006. Springer, Heidelberg, Germany.

[DFK$^+$06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany. doi: 10.1007/11681878_15

[DGH$^+$21] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Pasta: A case for hybrid homomorphic encryption. Cryptology ePrint Archive, Report 2021/731, 2021. `https://eprint.iacr.org/2021/731`.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-46800-5_24

[DPR16] Ivan Damgård, Antigoni Polychroniadou, and Vanishree Rao. Adaptively secure multi-party computation from LWE (via equivocal FHE). In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 208–233, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-49387-8_9

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-32009-5_38

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. doi: 10.1007/978-3-319-96881-0_2

[FMRV22] Thibauld Feneuil, Jules Maire, Matthieu Rivain, and Damien Vergnaud. Zero-knowledge protocols for the subset sum problem from MPC-in-the-head with rejection. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part II*, volume 13792 of *Lecture Notes in Computer Science*, pages 371–402, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany. doi: 10.1007/978-3-031-22966-4_13

[FR22] Thibauld Feneuil and Matthieu Rivain. Threshold linear secret sharing to the rescue of MPC-in-the-head. Cryptology ePrint Archive, Report 2022/1407, 2022. `https://eprint.iacr.org/2022/1407`.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-54242-8_4

[GMPP16] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 448–476, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-49896-5_16

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. doi: 10.1007/978-3-642-40041-4_5

[KKL$^+$22] Taechan Kim, Hyesun Kwak, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition. Cryptology ePrint Archive, Report 2022/347, 2022. `https://eprint.iacr.org/2022/347`.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 525–537, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi: 10.1145/3243734.3243805

[KLP18] Eunkyung Kim, Hyang-Sook Lee, and Jeongeun Park. Towards round-optimal secure multiparty computations: Multikey FHE without a CRS. In Willy Susilo and Guomin Yang, editors, *ACISP 18: 23rd Australasian Conference on Information Security and Privacy*, volume 10946 of *Lecture Notes in Computer Science*, pages 101–113, Wollongong, NSW, Australia, July 11–13, 2018. Springer, Heidelberg, Germany. doi: 10.1007/978-3-319-93638-3_7

[KMS22] Hyesun Kwak, Seonhong Min, and Yongsoo Song. Towards practical multi-key TFHE: Parallelizable, key-compatible, quasi-linear complexity. Cryptology ePrint Archive, Report 2022/1460, 2022. `https://eprint.iacr.org/2022/1460`.

[KÖA23] Jakub Klemsa, Melek Önen, and Yavuz Akın. A practical TFHE-based multi-key homomorphic encryption with linear complexity and low noise growth. Cryptology ePrint Archive, Report 2023/065, 2023. `https://eprint.iacr.org/2023/065`.

[Lib23] Benoit Libert. Vector commitments with short proofs of smallness. Cryptology ePrint Archive, Paper 2023/800, 2023. `https://eprint.iacr.org/2023/800`.

[LMK$^+$22] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. Cryptology ePrint Archive, Report 2022/198, 2022. `https://eprint.iacr.org/2022/198`.

[LP19] Hyang-Sook Lee and Jeongeun Park. On the security of multikey homomorphic encryption. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *Lecture Notes in Computer Science*, pages 236–251, Oxford, UK, December 16–18, 2019. Springer, Heidelberg, Germany. doi: 10.1007/978-3-030-35199-1_12

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing*, pages 1219–1234, New York, NY, USA, May 19–22, 2012. ACM Press. doi: 10.1145/2213977.2214086

[MTBH21] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies*, 2021(4):291–311, October 2021. doi: 10.2478/popets-2021-0071

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 735–763, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-49896-5_26

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 217–238, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. doi: 10.1007/978-3-662-53644-5_9

[RST+22] Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for SPDZ. *Journal of Cryptology*, 35(1):5, January 2022. doi: 10.1007/s00145-021-09416-w

[YKHK18] Satoshi Yasuda, Yoshihiro Koseki, Ryo Hiromasa, and Yutaka Kawai. Multi-key homomorphic proxy re-encryption. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *ISC 2018: 21st International Conference on Information Security*, volume 11060 of *Lecture Notes in Computer Science*, pages 328–346, Guildford, UK, September 9–12, 2018. Springer, Heidelberg, Germany. doi: 10.1007/978-3-319-99136-8_18

[ZDH20] Ruiyu Zhu, Changchang Ding, and Yan Huang. Practical MPC+FHE with applications in secure multi-PartyNeural network evaluation. Cryptology ePrint Archive, Report 2020/550, 2020. `https://eprint.iacr.org/2020/550`.