Mixed-Technique, Maliciously-Secure, and Composed Multi-Party Computations

Erik-Oliver Blass
Airbus
Munich, Germany
erik-oliver.blass@airbus.com

Florian Kerschbaum
University of Waterloo
Waterloo, Canada
florian.kerschbaum@uwaterloo.ca

ABSTRACT

Efficient multi-party protocols are commonly composed of different sub-protocols, combining techniques such as homomorphic encryption, secret or Boolean sharing, and garbled circuits. To ensure security of the composed protocol against malicious adversaries, one needs to prove in zero-knowledge that conversions between individual techniques are correct. However, efficient ZK proofs for conversion between fully homomorphic encryption and garbled circuits are still an open problem. In this paper, we design new efficient proofs and apply them to a new class of multi-party protocols which themselves are composed out of two-party protocols. We integrate both types of compositions, compositions of fully homomorphic encryption with garbled circuits and compositions of multi-party protocols from two-party protocols. As a result, we can construct communication-efficient protocols for special problems with malicious security. To show the usefulness of this approach, we give an example scheme for private set analytics, i.e., private set disjointness. This scheme enjoys lower communication complexity than a solution based on generic multi-party protocols and lower computation cost than fully homomorphic encryption. So, our design is more suitable for deployments in wide-area networks, such as the Internet, with many participants or problems with circuits of moderate or high multiplicative depth.

1 INTRODUCTION

Whereas two-party secure computations are deployed in practice [62], designing and deploying practical secure multi-party computation is still an open challenge. Communication latency is a typical bottleneck for many multi-round protocols, and in response constant-round multi-party computations [31, 40, 41] based on Beaver et al.'s technique [3] have been designed. Their deployment is lacking due to challenges from implementation complexity, communication bandwidth, and memory requirements. To address these challenges, protocols using fully-homomorphic encryption (FHE) [9, 21] and dual execution can be used. Yet, designing efficient homomorphic encryption schemes (for arithmetic circuits) is also an open challenge. Circuits with high multiplicative depth, the reason for a high number of rounds in many multi-party computation protocols, imply large computation costs.

In this paper, we present a design alternative. We consider multi-party computations that can at least partially be decomposed into two-party computations (2PCs). We start by combining 2PCs using garbled circuits with FHE. The idea of our mixed-technique protocols is to exploit advantages of each technique, for example, binary vs. arithmetic circuits, typical in application domains such as machine learning [10, 18, 28, 45]. More specifically, we show how to convert between outputs of garbled circuits and FHE ciphertexts using efficient zero knowledge proofs for fully malicious security. This is a conversion which has not been considered in previous work. A combined multi-party protocol can start, e.g., with a number of 2PCs and then continues using FHE evaluation. The first phase of 2PC reduces multiplicative depth for the following FHE evaluation phase, but remains small enough to have low communication complexity. Such a combined protocol keeps a constant number of rounds and is still secure in the malicious model. Due to their lower communication requirements, combined protocols have the potential for deployment in wide area networks.

The composition of 2PC protocols into a multi-party protocol can take many forms. In order to demonstrate the advantages of our constructions, we design and investigate a combined protocol for multi-party set analytics. This protocol follows a star topology of communication where each party P_i engages in 2PC with a central party P_1. Driven by the use case of sharing Indicators of Compromise (IoCs), where multiple parties try to determine whether they have been subject to a common attack, we design a maliciously-secure protocol which determines whether the multi-party set intersection is empty. A non-empty intersection would be grounds for further investigation. With each party's set holding n elements, our set disjointness protocol runs in 9 rounds, needs $O(n)$ broadcasts, and has a message complexity linear in the number of comparisons required to compare all parties' inputs.

As we will see, this example of set analytics can be realized with a star topology of 2PC, as equality is symmetric and transitive. Other examples, e.g., larger-than comparison, are not. Yet, following a full pair-wise communication pattern, another example, where our constructions can be applied, is ranking the inputs of all parties. For rank computation as used in, e.g., auctions [7], our constructions lead to a protocol with stronger security compared to recent results [7], yet still maintains a constant number of rounds.

In compositions of multi-party protocols from two-party protocols an important application of our conversions can be used. When there are multiple two-party protocols by one party within a composed protocol, this one party may need to commit to its input before all two-party protocols and prove that all two-party protocols use the same input by opening the commitment in the 2PC. The common technique to implement this is to use hash-based commitments and verify the hash in the secure computation. This requires about 22000 AND gates for each 256 input bits using SHA2 [22]. Our construction can be used as an alternative. We use homomorphic ciphertexts of a shared, fixed key as commitments, and verification takes only 80 AND gates per input bit (at a statistical security level of 40 bit).

In summary, our major contributions are

- Efficient zero-knowledge proofs for the conversion between garbled circuit outputs and homomorphic encryption.
- A construction for mixed-technique MPC that has a constant number of rounds and a low communication complexity, yet is secure in the malicious model. This construction is exemplified using a multi-party protocol for set disjointness.

2 CONVERSION BETWEEN 2PC AND HOMOMORPHIC ENCRYPTION

To simplify exposition, we start by giving an overview about our conversion for the special case of $d = 2$ parties. Later in Section 3.5.1, we present an extension for any $d \geq 2$ parties.

Parties P_1 and P_2 want to jointly compute function $F(I_1, I_2) = O$ on their respective input bit strings I_1 and I_2 to receive output string $O = (o_1, \cdots, o_N)$. For security reasons, P_1 should only learn some subset of bit string O, but nothing else (for example not P_2's input). Similarly, P_2 should only learn the other bits of O, but nothing else. To enable secure computation of F, parties can revert to two standard approaches. Parties could express F as a Boolean circuit and evaluate this circuit using maliciously-secure two-party garbled circuit computation (2PC). Alternatively, parties express F as an arithmetic circuit, compute a shared private key of a fully homomorphic encryption (FHE), and encrypt their inputs with the corresponding public-key. Parties then evaluate the circuit homomorphically and jointly decrypt the final result such that each party only learns their output bits.

Yet, each of the two approaches comes with performance issues. One the one hand, FHE evaluation of arithmetic circuits with large multiplicative depth is computationally expensive. One the other hand, evaluating Boolean circuits with 2PC for large circuits is expensive regarding the amount of communication.

So, a third alternative and the focus of this paper is for parties to evaluate F using a *mix* of both techniques. Parties evaluate F as a circuit decomposed into a sequence of sub-circuits $F(I_1, I_2) = (C_1 \circ \cdots \circ C_m)(I_1, I_2)$. Some sub-circuits C_i are Boolean, while others are arithmetic. Parties agree that Boolean sub-circuits of function F will be evaluated using garbled circuit 2PC, and arithmetic sub-circuits of F will be evaluated using FHE. The goal of such a mixed-techniques approach is to optimize overall performance by reducing multiplicative depth of FHE circuits and communication complexity of 2PC circuits. For clarity, we now denote Boolean (sub-)circuits C_i by C_i^{Bool} and arithmetic (sub-)circuits C_i by C_i^{Arith} in this paper. Assume that P_1 and P_2 have initially computed a public and private key pair for a homomorphic encryption Enc, where the private key is shared among both parties.

Technical Challenges: While the scheme above seems straightforward, it implies several technical challenges. Output from one mode of circuit evaluation must be converted into input of the other mode. In the clear, converting output between Boolean and arithmetic circuits is simple, as parties rely on the standard convention that a Boolean True equals 1, and False equals 0.

Yet, achieving malicious security for conversion is difficult. For example, let P_1 be the garbler and P_2 the evaluator during 2PC evaluation of a simple Boolean sub-circuit C_i^{Bool} with two input and two output bits $(x, y) = C_i^{\mathsf{Bool}}(a, b)$. Evaluator P_2 receives both output bits x, y and must convert them into correct homomorphic encryptions $\mathsf{Enc}(x)$ and $\mathsf{Enc}(y)$. This is hard to achieve against malicious adversaries. As P_2 could be malicious, P_2 must prove to P_1 that ciphertexts $\mathsf{Enc}(x)$ and $\mathsf{Enc}(y)$ are correctly encrypting outputs x and y received during 2PC. Worse, P_2 should not even learn x and y, as they are an intermediate result of C's evaluation or maybe output bits for P_1. Instead, P_2 should receive related information during 2PC which then allows P_2 to indirectly generate homomorphic encryptions $\mathsf{Enc}(x)$ and $\mathsf{Enc}(y)$. Again, we stress that simply implementing homomorphic encryption Enc inside a 2PC circuit is too expensive.

Similarly, we need to convert FHE ciphertexts output by arithmetic circuits C_i^{Arith} into input for 2PC garbled circuits with malicious security. Moreover, if P_1 and P_2's 2PC computation was part of a larger MPC computation involving $d \geq 2$ parties, we also need to consider the case where both are malicious, and they must prove to all d parties that their encrypted shares are correct. Finally, in that case the private key is shared among all d parties, which impedes easy Zero Knowledge proofs.

Remarks: To securely evaluate Boolean sub-circuits C_i^{Bool}, we assume existence of any maliciously secure 2PC scheme as a building block. Several different approaches exist which achieve maliciously secure 2PC in practice, see [38, 39, 48, 60] for an overview.

For secure evaluation of arithmetic sub-circuits C_i^{Arith}, any FHE scheme could serve as building block. FHE is maliciously secure by default, as long as parties evaluate the same circuit on the same ciphertexts, respectively. However, our conversion requires the FHE scheme to also support distributed key generation and certain ZK proofs. There exist several efficient lattice-based FHE schemes with support for both [5, 6, 8, 14, 15, 46, 57], and there are even efficient schemes which allow proving general, arbitrary ZK statements in addition to distributed key generation [2]. While describing details of our techniques, we use any of these as an underlying building block, e.g., the one by Asharov et al. [2].

2.1 Solution High-Level Overview

There are two different cases for conversion we have to consider in a mixed-technique setting. A first case is that parties must convert output bits $(o_{i,1}, \ldots, o_{i,n}) = C_i^{\mathsf{Bool}}(I_{i,1}, I_{i,2})$ from 2PC evaluation of circuit C_i^{Bool} on input strings $I_{i,1}$ and $I_{i,2}$ into n homomorphic encryptions $\mathsf{Enc}(o_{i,j})$. Knowing encryptions $\mathsf{Enc}(o_{i,j})$, each party can then evaluate the subsequent arithmetic circuit C_{i+1}^{Arith}, respectively.

The second case is converting a sequence of ciphertexts $\mathsf{Enc}(b_i)$, homomorphic encryptions of bits b_i (or integers, see Section 3.5.3) into input for a 2PC Boolean circuit evaluation. That is, both parties have evaluated an arithmetic sub-circuit C_i^{Arith} and computed ciphertexts $\mathsf{Enc}(b_i)$, respectively. These ciphertexts must now be converted into input for 2PC evaluation of next sub-circuit C_{i+1}^{Bool}.

Parties will also need to securely convert both parties' plain input into either FHE encryptions or 2PC inputs. Yet, that part is trivial: if the first sub-circuit is an arithmetic circuit, a party sends homomorphic encryptions of each input bit. If the first circuit is Boolean, we rely on whatever technique the underlying maliciously secure 2PC offers. Finally, at the end of the last circuit evaluation, FHE ciphertexts or 2PC output has to be decrypted. Again, this is fairly simple, and we skip details for now.

Therefore, we only concentrate on the first two cases of converting 2PC output to FHE input and FHE output to 2PC input.

2.1.1 Main idea. Our conversions focuses on Boolean sub-circuits C_i^{Bool}. We design mechanisms which either (I) convert 2PC output of C_i^{Bool} to FHE ciphertexts serving as input to C_{i+1}^{Arith} or (II) to convert FHE ciphertexts coming from C_{i-1}^{Arith} into input to C_i^{Bool}. Actual evaluation of circuits is then secure by definition. For Boolean sub-circuits, we rely on standard maliciously-secure 2PC. For arithmetic sub-circuits, both parties evaluate FHE ciphertexts on their own. A honest party will automatically compute correct output ciphertexts as long as input ciphertexts are correct.

Each of our two conversions first modifies C_i^{Bool} and evaluates the modified circuit using three new cryptographic building blocks which we call ZK Protocol (1), ZK Protocol (2), and ZK Protocol (3). Each ZK Protocol takes as input a Boolean circuit and P_1's and P_2's input bits. ZK Protocol (1) and ZK Protocol (2) also take FHE ciphertexts as inputs. Each ZK Protocol again modifies the input circuit internally, 2PC-evaluates the modified version, and outputs 2PC output together with a ZK proof which proves certain relations between input and output in zero knowledge. As ZK Protocols are general, their interesting property is to be stackable, i.e., they can be combined with each other. Their internal circuit modification schemes will be merged, and only ZK proofs enclosing circuit modification have to be adopted, which is rather mechanical.

In conclusion, we design conversion schemes combining multiple 2PC circuit modification techniques with efficient ZK proofs. Together, modifications and proofs prove correctness of output conversion between outputs of 2PC and FHE circuit evaluation.

2.1.2 ZK Protocols. Let γ be any Boolean circuit defined by its input and output bits as $(\omega_1, \ldots, \omega_n) = \gamma((\iota_{1,1}, \ldots, \iota_{1,\ell_1}), (\iota_{2,1}, \ldots, \iota_{2,\ell_2}))$. Parties P_1 and P_2 want to evaluate this circuit with 2PC. Bits $\iota_{1,i}$ are inputs of P_1. Bits $\iota_{2,i}$ are inputs of P_2, and ω_i will be output bits known to P_2. From a high-level perspective, our three ZK Protocols implement:

ZK Protocol (1): P_1 sends homomorphic ciphertexts $c_{1,i} \leftarrow \mathsf{Enc}(\iota_{1,i})$, encrypting their input bits $\iota_{1,i}$ to P_2. Circuit γ is evaluated, and P_2 receives output. P_1 proves in ZK to P_2 that c_i encrypts $\iota_{1,i}$, used during 2PC evaluation of γ.

ZK Protocol (2): P_2 sends homomorphic ciphertexts $c_{2,i} \leftarrow \mathsf{Enc}(\iota_{2,i})$, encrypting their input bits $\iota_{2,i}$ to P_1. Circuit γ is evaluated, and P_2 receives output. P_2 proves in ZK to P_1 that c_i encrypts $\iota_{2,i}$, used during 2PC evaluation of γ.

ZK Protocol (3): Circuit γ is evaluated, and P_2 receives output ω_i. Party P_2 sends homomorphic ciphertext $c_{\omega,i} \leftarrow \mathsf{Enc}(\omega_i)$ and proves in ZK to P_1 that $c_{\omega,i}$ really encrypts ω_i received during 2PC evaluation to P_1.

Note the different notation we use for describing circuits. Boolean sub-circuits of function F are written as C_i^{Bool} with input strings I_1, I_2 and output bits o_i. To avoid confusion, Boolean circuits we use within our ZK Protocol building blocks are written with Greek letters, i.e., γ.

We describe the intuition behind our ZK Protocols later, so assume them as given. Now, we present an overview on how we use these building blocks together with additional circuit modifications during conversions.

2.1.3 Conversion from FHE to 2PC. Let $c_1 \leftarrow \mathsf{Enc}(b_1), \ldots, c_\ell \leftarrow \mathsf{Enc}(b_\ell)$ be FHE ciphertexts of bits b_i. Parties P_1 and P_2 know the c_i, but not the b_i, and they want to use the b_i as input in Boolean sub-circuit evaluation, i.e., they want to securely evaluate Boolean circuit $(o_1, \ldots, o_n) = C_j^{\mathsf{Bool}}(b_1, \ldots, b_\ell)$. The idea is that, for each c_i, P_1 randomly chooses bit s_i and sends $\mathsf{Enc}(s_i)$ to P_2. As both parties know c_i, they both homomorphically compute $\mathsf{Enc}(s_i \oplus b_i)$ and jointly decrypt such that only P_2 receives plaintext bit s'_i with $s'_i = b_i \oplus s_i$. As a result, both parties know shares of each bit b_i.

Instead of securely evaluating Boolean circuit C_j^{Bool}, parties securely evaluate a modification of it which we call $\gamma_{\mathsf{Share},j}$. Circuit $\gamma_{\mathsf{Share},j}$ uses C_j^{Bool} internally as a subroutine. With input bits s_i coming from P_1 and s'_i from P_2, $\gamma_{\mathsf{Share},j}$ computes $b_i = s_i \oplus s'_i$ and evaluates C_j^{Bool} with the b_i as input. The output of $\gamma_{\mathsf{Share},j}$, only received by P_2, is the output of circuit C_j^{Bool}.

This is not yet secure: P_1 must also prove to P_2 that they have used correct s_i matching $c_i = \mathsf{Enc}(s_i)$. So, they use ZK Protocol (1) on top of the evaluation of $\gamma_{\mathsf{Share},j}$.

This is still not secure, as P_2 must prove to P_1 that they have used correct s'_i matching $\mathsf{Enc}(b_i \oplus s_i)$. So, they use ZK Protocol (2) also on top of the evaluation of $\gamma_{\mathsf{Share},j}$. We realize the security of this conversion from FHE to 2PC by combining ZK Protocol (1) and (2).

If C_j^{Bool} is not the last circuit, we need to additionally encrypt the o_i to homomorphic encryptions $\mathsf{Enc}(o_i)$ without P_1 or P_2 learning o_i. We securely encrypt the o_i by combining our scheme above with the 2PC to FHE conversion described next.

2.1.4 Conversion from 2PC to FHE. Let $(o_1, \ldots, o_n) = C_j^{\mathsf{Bool}}(I_1, I_2)$ be a Boolean sub-circuit which parties want to evaluate, and output $(o_1, \ldots, o_n)$ should be converted into FHE ciphertexts.

First, note that C_j^{Bool} can be either the first sub-circuit C_1 of function F, so bit strings $I_1 = (i_{1,1}, \ldots, i_{1,\ell_1}), I_2 = (i_{2,1}, \ldots, i_{2,\ell_2})$ are plain input values of P_1 and P_2, or C_j^{Bool} is a γ_{Share} circuit introduced above. In that case, P_1's input I_i and P_2's input I_2 are (plain) shares of bits b_i. So in any case, input bit $(i_{1,1}, \ldots, i_{1,\ell_1})$ are known to P_1, and $(i_{2,1}, \ldots, i_{2,\ell_2})$ are known to P_2.

The goal of the conversion is that, during 2PC evaluation of C_j^{Bool}, neither P_1 nor P_2 learns output $(o_1, \ldots, o_n)$, but instead correct FHE encryptions $c_i \leftarrow \mathsf{Enc}(o_i)$ are generated.

The idea behind 2PC to FHE conversion is essentially the opposite of the sharing approach from above. First, P_1 selects for each *output* bit o_i a random share bit s_i, homomorphically encrypts it and sends resulting ciphertext $\mathsf{Enc}(s_i)$ to P_2.

Parties do not evaluate C_j^{Bool} but a circuit $\gamma_{\mathsf{Share}'_j}$ which uses C_j^{Bool} internally as a sub-routine. Circuit $\gamma_{\mathsf{Share}'_j}$ evaluates C_j^{Bool} on inputs I_i, I_2 and outputs, for each output bit o_i, the XOR $\omega_i = s_i \oplus o_i$ to P_2. Party P_2 homomorphically encrypts ω_i and sends $\mathsf{Enc}(\omega_i)$ to P_1. To make the above secure, we run a combination of ZK Protocol 1 and ZK Protocol 3 on $\gamma_{\mathsf{Share}'_j}$.

As a consequence each party knows a share of o_i. Party P_1 knows s_i, P_2 knows ω_i, and both know encryptions of shares. To conclude conversion, both parties homomorphically compute an encryption of o_i out of the shares.

2.2 ZK Protocols Intuition

Before finally turning toward technical details, we briefly describe the main idea behind ZK Protocols. We describe our protocols making black-box use of 2PC. This ensures that our protocols are compatible with any 2PC protocols secure against malicious adversaries and includes the use of commitments in the 2PC. In Section 3.5.2 we show how to replace those commitments with a white-box use of wire labels in garbled circuits.

2.2.1 *ZK Protocol (1).*

Party P_1 has to prove to P_2 that homomorphic encryption $c_{1,i} \leftarrow \mathsf{Enc}(\iota_{1,i})$ really encrypts input bit $\iota_{1,i}$ used during 2PC evaluation of some circuit γ. To prove this, we set up a three move ZK proof as a basis.

First, P_1 selects a random *masking* bit μ_i and sends both $c_{1,i}$ and $m_i \leftarrow \mathsf{Enc}(\mu_i)$ to P_2. At the same time, P_2 selects a random *choice* bit σ_i. Then, instead of securely evaluating circuit γ, both parties use maliciously-secure 2PC and evaluate a slight extension of γ called $\gamma^{(1)}$ which internally computes γ as a sub-routine. Party P_1 is the garbler and P_2 the evaluator. Circuit $\gamma^{(1)}$ takes μ_i and σ_i as additional inputs. In addition to outputting the same bits as γ, it also outputs bit $t_i = \iota_{1,i} \oplus \mu_i$ (if $\sigma_i = 0$) or $t_i = \mu_i$ (if $\sigma_i = 1$) to P_2.

After 2PC, P_2 reveals their choice σ_i. If $\sigma_i = 0$, then P_1 proves in ZK that the homomorphic XOR of ciphertexts $c_{1,i}$ and m_i to $\mathsf{Enc}(\iota_{1,i} \oplus \mu_i)$ really encrypts $t_i = \iota_{1,i} \oplus \mu_i$. If $\sigma_i = 1$, then P_1 proves that m_i encrypts $t_i = \mu_i$.

Roughly speaking, P_2 verifies whether the output received from γ matches homomorphic encryptions. The security argument is that P_1 does not know σ_i in advance. Party P_1 is forced to encrypt μ_i correctly while at the same time the homomorphic XOR of encryptions of μ_i and $\iota_{1,i}$ must match. P_1 can cheat in this ZK proof by having $c_{1,i}$ encrypt $1 \oplus \iota_{1,i}$ instead of $\iota_{1,i}$ with probability $\frac{1}{2}$. Thus for some statistical security parameter λ, we repeat the above a total of λ times, where P_2 supplies λ random choices $\sigma_{i,j}$, and P_1 supplies λ encryptions of masking bits $\mathsf{Enc}(\mu_{i,j})$. Therewith, chances of P_1 cheating in the proof are reduced to a value negligible in λ. We run this ZK proof for each input bit $\iota_{1,i}$ and corresponding ciphertext $c_{1,i}$ in parallel.

While the above ZK proof (intuitively) solves the problem of forcing P_1 to use a $\iota_{1,i}$ in γ which is consistent with $\mathsf{Enc}(\iota_{1,i})$, it introduces additional, yet much simpler challenges. For example, against a malicious P_2, we will have to ensure that $\sigma_{i,j}$ used in $\gamma^{(1)}$ are the same $\sigma_{i,j}$ as revealed by P_2 in the ZK proof. For ease of exposition, we address these challenges later during formal description.

2.2.2 *ZK Protocol (2).*

This is essentially the inverse of ZK Protocol (1). Now, it is P_2 who selects a random masking bit μ'_i for each input bit $\iota_{2,i}$ and sends $c_{2,i} \leftarrow \mathsf{Enc}(\iota_{2,i}), m'_i \leftarrow \mathsf{Enc}(\mu'_i)$ to P_1. At the same time, P_1 selects random choice bit σ'_i.

With P_1 being the garbler and P_2 the evaluator, parties use maliciously-secure 2PC and evaluate a circuit $\gamma^{(2)}$ which is similar to $\gamma^{(1)}$. The difference is that, here, σ'_i is an additional input for P_1 and μ'_i an input for P_2. Now depending on σ'_i, P_2 would receive either $t'_i = \iota_{2,1} \oplus \mu'_i$ or $t'_i = \mu'_i$. As t'_i is an output for P_1, and P_2 must not learn it, we use the same trick to blind t'_i from above: P_1 uses another random bit b_i as input into $\gamma^{(2)}$, and P_2 does not receive t'_i, but $t'_i \oplus b_i$ which it can forward to P_1.

2.2.3 *ZK Protocol (3).*

In this protocol, P_2 will send a ciphertext $c_{\omega,i} \leftarrow \mathsf{Enc}(\omega_i)$ to P_1 and prove that $c_{\omega,i}$ is indeed a homomorphic encryption of bit ω_i output by γ.

The idea is as follows. Before 2PC, P_1 selects for an output bit ω_i two random bit strings $v_{0,1} \ldots v_{0,\lambda}$ and $v_{1,1} \ldots v_{1,\lambda}$ and sets $V_0 = 0||v_{0,1} \ldots v_{0,\lambda}, V_1 = 1||v_{1,1} \ldots v_{1,\lambda}$. Here, "||" denotes concatenation, and λ is a statistical security parameter. Then, P_1 encrypts and sends ciphertexts $\Gamma_0 = \mathsf{Enc}(V_0)$ and $\Gamma_1 = \mathsf{Enc}(V_1)$ to P_2.

Instead of evaluating γ, parties again use maliciously-secure 2PC (P_1 garbler, P_2 evaluator) and evaluate a slightly modified circuit $\gamma^{(3)}$ of γ. Circuit $\gamma^{(3)}$ internally computes γ as a subroutine. Input to $\gamma^{(3)}$ is $\iota_{1,i}$ and in addition the $2 \cdot \lambda$ bits $v_{i,j}$ for P_1 and $\iota_{2,i}$ for P_2. Circuit $\gamma^{(3)}$ does not output ω_i to P_2, but instead outputs V_{ω_i} to P_2, i.e., either bit string V_0 or bit string V_1.

It is important to note that the first bit of strings V_0, V_1 is output bit ω_i. Therewith, Γ_{ω_i} encrypts a bit string, where the first bit represents P_2's output bit ω_i. For the case $\omega_i = 0$, it is Γ_0 which encrypts a string where the first bit is 0. For the case $\omega_i = 1$, it is Γ_1 which encrypts a string with first bit equal 1. So, after evaluating $\gamma^{(3)}$, P_2 gets ω_i and a length λ bit string $(v_{\omega_i,1}, \ldots, v_{\omega_i,\lambda})$.

The trick is now that P_2 can prove in ZK to P_1 that it knows a string V_{ω_i} which is *either* V_0 *or* V_1 and which matches encryption $c_{\omega,i}$. Recall that the private key for homomorphic encryption Enc is shared between P_1 and P_2, so none of the two parties can decrypt a ciphertext alone. After evaluating $\gamma^{(3)}$, party P_2 sends $\lambda + 1$ ciphertexts $c_{\omega,i} \leftarrow \mathsf{Enc}(\omega_i), \mathsf{Enc}(v_{\omega_i,1}, \ldots, \mathsf{Enc}(v_{\omega_i,\lambda}))$ to P_1. Both parties use these ciphertexts to homomorphically generate $\Gamma_2 = \mathsf{Enc}(V_{\omega_i})$, i.e., an encryption of the concatenation of P_2's $\lambda + 1$ bits V_{ω_i}. How is rather straightforward, but we skip details for now.

As both parties know Γ_0 and Γ_1, they can both homomorphically compute $\Delta_0 = \mathsf{Enc}(V_{\omega_i} - V_0)$ and $\Delta_1 = \mathsf{Enc}(V_{\omega_i} - V_1)$. Observe that, if V_{ω_i} is either V_0 or V_1, then one of Δ_0, Δ_1 encrypts a 0. Consequently, P_2 proves to P_1 in ZK that either Δ_0 or Δ_1 is an encryption of 0. Again, we skip details for now and only summarize the idea: P_2 blinds both Δ_0 and Δ_1 homomorphically with the same random factor, shuffles blinded ciphertexts, proves correctness of both using the same random factor and the shuffle in ZK, and together with P_1 decrypts resulting ciphertexts. If one of the two decryptions is 0, then P_1 accepts that $c_{\omega,i}$ encrypts ω_i without learning ω_i.

We run the above technique for each output bit ω_i in parallel.

3 TECHNICAL DETAILS

For their input bit strings $I_1, I_2 \in \{0,1\}^*$ and function F, parties P_1 and P_2 want to compute $O = F(I_1, I_2), O \in \{0,1\}^*$. Function F is represented as a circuit composition of Boolean and arithmetic sub-circuits $F = (C_m \circ \cdots \circ C_1)$. Observe that if the i^{th} sub-circuit is Boolean, then the $i+1^{\text{th}}$ is arithmetic and the other way around. We now turn toward technical details on how we enable maliciously-secure mixed-technique evaluation of sub-circuits. That is, we show how to convert 2PC evaluation output of a Boolean sub-circuit C_i^{Bool} into input for a following arithmetic sub-circuit C_{i+1}^{Arith} for FHE evaluation and the other way around.

2PC output bits for P_1: In a typical garbled circuit evaluation of C_i, only P_2 receives output, i.e., bits o_j. If a specific bit o_j is a secret output bit for P_1, then a standard trick is denying P_2 to open the last wire label for o_j and forwarding the label to P_1. As P_1 knows both possible labels for o_j, they can recover bit o_j. Therewith you can also make sure that P_1 receives the correct output bit o'_j from P_2, i.e., ensure authenticity [4]. We silently rely on this trick for secure computation of all of P_1's plain output bits for the rest of the paper.

3.1 Notation

Let Commit denote a computationally hiding and binding commitment scheme. For some bit string $B \in \{0,1\}^*$, computational security parameter λ', and randomness $R \in \{0,1\}^{\lambda'}$, $\mathsf{Commit}(B,R)$ outputs a commitment Com.

Encryption Enc over plaintext space M is fully (or somewhat) homomorphic. Both parties have already set up a key pair, where the public key is known to both parties, but the private key is shared. For homomorphic operations on ciphertexts, we use the intuitive notation of "+" for homomorphic addition, "·" for scalar multiplication, and $\oplus$ for homomorphic XOR. So for example, if x and y are from M, then $\mathsf{Dec}(\mathsf{Enc}(x) + \mathsf{Enc}(y)) = x + y$. During conversion, we will randomly select scalars from $\mathbb{Z}_p$, where p is a prime of λ bits.

Let Π be the set of two single bit permutations $\pi : \{0,1\} \longrightarrow \{0,1\}$. That is, $\Pi = \{\pi_0, \pi_1\}$ with $\pi_0(x) = x$ and $\pi_1(x) = 1 - x$.

3.2 ZK Protocols

Let $(\omega_1, \ldots, \omega_n) = \gamma((\iota_{1,1}, \ldots, \iota_{1,\ell_1}), (\iota_{2,1}, \ldots, \iota_{2,\ell_2}))$ be any Boolean circuit which parties P_1 and P_2 want to evaluate using maliciously secure 2PC. Bits $\iota_{1,i}$ are P_1's input, and bits $\iota_{2,i}$ are P_2's input.

3.2.1 ZK Protocol (1). In this protocol, P_1 proves to P_2 that homomorphic ciphertexts $c_{1,i} \leftarrow \mathsf{Enc}(\iota_{1,i})$ encrypt all of P_1's input bits $\iota_{i,i}$ used during a 2PC evaluation of γ. Assume that P_1 has already sent the $c_{1,i}$ to P_2.

The protocol is depicted in Figure 1 and consists of two core building blocks: first, parties evaluate a modification of circuit γ which we call $\gamma^{(1)}$. We define circuit $\gamma^{(1)}$ by specifying its input and output in Figure 2. The second building block is an actual three move ZK proof which encompasses $\gamma^{(1)}$.

Observe that P_1 verifies P_2's commitments $\mathsf{Com}_{i,j}$ to choice bits $\sigma_{i,j}$ twice, once in circuit $\gamma^{(1)}$, and once after 2PC while proving that either masking bits $\mu_{i,j}$ is encrypted in $m_{i,j}$ or input bit $\iota_{1,i} \oplus \mu_{i,j}$ is encrypted in $c_{1,i} \oplus m_{i,j}$. Therewith, P_1 is convinced that P_2 uses the same choice bits consistently. We remind the reader that in Section 3.5.2 we replace commitments by the white-box use of wire labels in garbled circuits. If $\sigma_{i,j} = 0$, then P_1 and P_2 homomorphically compute $\text{ciphertext}_{i,j} = \mathsf{Enc}(\iota_{1,i} \oplus \mu_{i,j})$ out of $c_{1,i}$ and $m_{i,j}$. If choice bit $\sigma_{i,j} = 1$, then both parties set $\text{ciphertext}_{i,j} = m_{i,j}$. Party P_1 then sends a ZK proof that $\text{ciphertext}_{i,j}$ really encrypts $t_{i,j}$ to P_2, e.g., by applying an efficient framework for ZK proofs [2].

Note the general structure of ZK Protocol (1), which is similar in the other two ZK Protocols. Each ZK Protocol comprises a circuit modification technique, here converting γ to $\gamma^{(1)}$, and a surrounding ZK proof. When we will combine ZK Protocols later, we merge circuit modifications, i.e., output of one ZK Protocol's circuit modification will be input into another. Only surrounding ZK proofs require adoption.

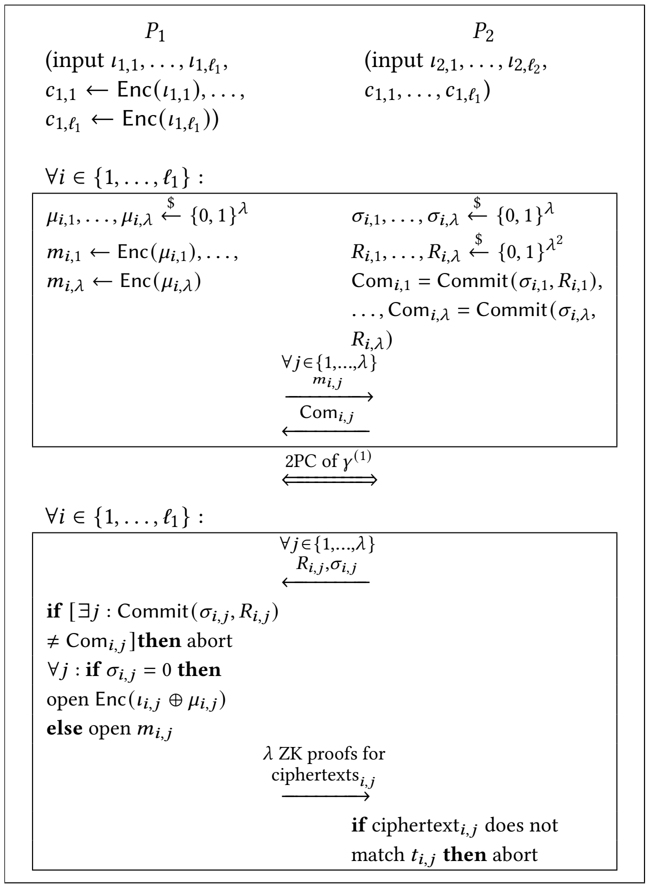

P_1
(input $\iota_{1,1}, \ldots, \iota_{1,\ell_1}$, $c_{1,1} \leftarrow \mathsf{Enc}(\iota_{1,1}), \ldots,$ $c_{1,\ell_1} \leftarrow \mathsf{Enc}(\iota_{1,\ell_1})$)
P_2
(input $\iota_{2,1}, \ldots, \iota_{2,\ell_2}$, $c_{1,1}, \ldots, c_{1,\ell_1}$)
$\forall i \in \{1, \ldots, \ell_1\}$:
$\mu_{i,1}, \ldots, \mu_{i,\lambda} \xleftarrow{\$} \{0,1\}^\lambda$
$m_{i,1} \leftarrow \mathsf{Enc}(\mu_{i,1}), \ldots,$ $m_{i,\lambda} \leftarrow \mathsf{Enc}(\mu_{i,\lambda})$
$\sigma_{i,1}, \ldots, \sigma_{i,\lambda} \xleftarrow{\$} \{0,1\}^\lambda$
$R_{i,1}, \ldots, R_{i,\lambda} \xleftarrow{\$} \{0,1\}^{\lambda^2}$
$\mathsf{Com}_{i,1} = \mathsf{Commit}(\sigma_{i,1}, R_{i,1}), \ldots, \mathsf{Com}_{i,\lambda} = \mathsf{Commit}(\sigma_{i,\lambda}, R_{i,\lambda})$
$\forall j \in \{1,\ldots,\lambda\}$
$\xrightarrow{m_{i,j}}$
$\xleftarrow{\mathsf{Com}_{i,j}}$
$\xLeftrightarrow{\text{2PC of } \gamma^{(1)}}$
$\forall i \in \{1, \ldots, \ell_1\}$:
$\forall j \in \{1,\ldots,\lambda\}$
$\xleftarrow{R_{i,j}, \sigma_{i,j}}$
if $[\exists j : \mathsf{Commit}(\sigma_{i,j}, R_{i,j}) \neq \mathsf{Com}_{i,j}]$ **then** abort
$\forall j$: **if** $\sigma_{i,j} = 0$ **then** open $\mathsf{Enc}(\iota_{i,j} \oplus \mu_{i,j})$
else open $m_{i,j}$
λ ZK proofs for $\text{ciphertexts}_{i,j}$
if $\text{ciphertext}_{i,j}$ does not match $t_{i,j}$ **then** abort

Figure 1: ZK Protocol (1) for circuit γ

3.2.2 ZK Protocol (2). This protocol reverses P_1's and P_2's roles in ZK Protocol (1). So, circuit $\gamma^{(2)}$ is similar to $\gamma^{(1)}$, with P_1 having choice bits (and randomness for commitments to them) as additional input, and P_2 has masking bits and commitments to choice bits as input. Also, the actual three-move protocol from ZK Protocol (1) is reversed, i.e., it is P_2 who starts by sending encryptions of input bits and masking bits. We omit further details to avoid repetition and refer to Figure 1.

3.2.3 ZK Protocol (3). In this protocol, party P_2 proves to P_1 that encryptions $c_{\omega,i} \leftarrow \mathsf{Enc}(\omega_i)$ are really encryptions of P_2's output bits ω_i. As ZK Protocol (3) is more involving, Figure 3 presents details of a slightly simpler version with a ZK proof which is only Honest-Verifier-Zero-Knowledge (HVZK). As part of ZK Protocol (3), P_1 and P_2 run 2PC on a modification of circuit γ called $\gamma^{(3)}$, defined in Figure 4.

Observe that P_1's additional inputs are $2 \cdot \lambda$ bits $v_{i,0,j}, v_{i,1,j}$, respectively, *for each* of the n output bits ω_i. Thus, there are also $\Gamma_{i,0,j}$ and $\Gamma_{i,1,j}$ for each ω_i, $\Gamma_{i,2,0} \leftarrow \mathsf{Enc}(\omega_1)$, λ-many $\Gamma_{i,2,j} \leftarrow \mathsf{Enc}(v_{i,\omega_i,j})$, and $\Delta_{i,0}$ and $\Delta_{i,1}$.

ZK Proof of 0: Figure 3 also comprises details for the ZK proof, where P_2 proves that either $\Delta_{i,0}$ or $\Delta_{i,1}$ encrypts a zero. In Figure 3, the blinded $\Delta_{i,0}$ and $\Delta_{i,1}$ are denoted by $\Delta'_{i,0}$ and $\Delta'_{i,1}$. In that proof, P_2 prepares sub-ZK proof "Scalar$_i$" which proves that $\Delta'_{i,0}, \Delta'_{i,1}$ are

Input to $\gamma^{(1)}$

P_1	P_2
$\iota_{1,1}, \ldots, \iota_{1,\ell_1}$, $1 \le i \le \ell_1 : [\mu_{i,1}, \ldots, \mu_{i,\lambda}$, $\mathsf{Com}_{i,1}, \ldots, \mathsf{Com}_{i,\lambda}]$	$\iota_{2,1}, \ldots, \iota_{2,\ell_2}$, $1 \le i \le \ell_1 : [\sigma_{i,1}, \ldots, \sigma_{i,\lambda}$, $R_{i,1}, \ldots, R_{i,\lambda}]$

Output of $\gamma^{(1)}$

```
1  if $\forall i, j, 1 \le i \le \ell_1, 1 \le j \le \lambda : \mathsf{Com}_{i,j} = \mathsf{Commit}(\sigma_{i,j}, R_{i,j})$
   then
2      $\alpha = 1$;
3      $(\omega_1, \ldots, \omega_n) = \gamma((\iota_{1,1}, \ldots, \iota_{1,\ell_1}), (\iota_{2,1}, \ldots, \iota_{2,\ell_2}))$;
4      for $i = 1$ to $\ell_1$ and $j = 1$ to $\lambda$ do
5          if $\sigma_{i,j} = 0$ then
6              $t_{i,j} = \iota_{1,i} \oplus \mu_{i,j}$;
7          else
8              $t_{i,j} = \mu_{i,j}$;
9  else
10     $\alpha = \omega_1 = \ldots = \omega_n = t_{1,1} = \ldots = t_{\ell_1,\lambda} = 0$;
11 output $\alpha, \omega_1, \ldots, \omega_n, t_{1,1}, \ldots, t_{\ell_1,\lambda}$;
```

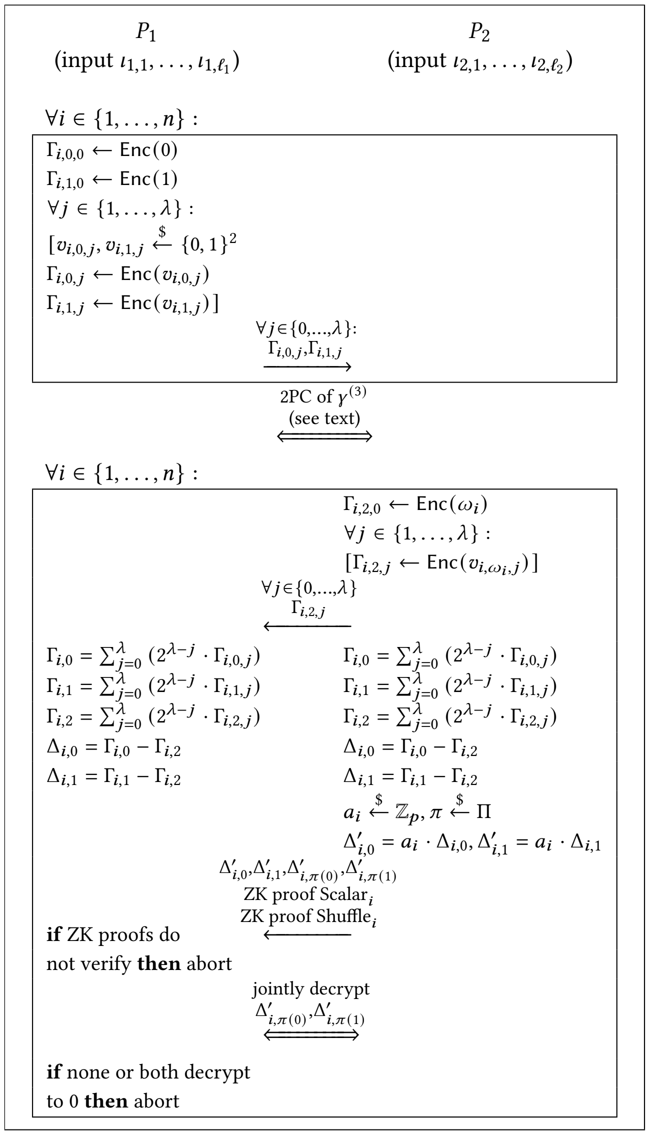
Figure 2: Definition of circuit $\gamma^{(1)}$

the result of multiplying $\Delta_{i,0}, \Delta_{i,1}$ by the same secret scalar a_i. Such a proof is typically uncomplicated with FHE, e.g., P_2 could simply publish the encryption of a_i, and P_1 computes $\Delta'_{i,0}, \Delta'_{i,1}$ themselves. As shown in Figure 3, P_2 completes the ZK proof by re-encrypting $\Delta'_{i,0}$ and $\Delta'_{i,1}$, choosing a random 1-bit permutation π from Π, and preparing ZK proof Shuffle$_i$ which proves that $(\Delta'_{i,\pi(0)}, \Delta'_{i,\pi(1)})$ is a random shuffle of $(\Delta'_{i,0}, \Delta'_{i,1})$. Proofs of two-element shuffles are also rather straightforward. For example, P_2 could encrypt a random bit to ciphertext β, send β to P_1, and prove that ciphertext $\beta - \beta^2$ encrypts a 0. Such a proof can be implemented by, e.g., reverting to an efficient general proof [2] or by opening randomness of ciphertext $\beta^2 - \beta$. Party P_1 then computes $\Delta'_{i,\pi(0)} = \beta \cdot \Delta'_{i,0} + (\mathsf{Enc}(1) - \beta) \cdot \Delta'_{i,1}$ and $\Delta'_{i,\pi(1)} = (\mathsf{Enc}(1) - \beta) \cdot \Delta'_{i,0} + \beta \cdot \Delta'_{i,1}$ themselves.

If P_1 successfully verifies proofs, parties jointly decrypt $\Delta'_{i,\pi(0)}$ and $\Delta'_{i,\pi(1)}$. Note that decryption must include a ZK proof by P_2 about correct (partial) decryption [2, 5, 8].

From HVZK to malicious security: A malicious verifier P_1 could cheat in ZK Protocol (3) described so far by using different bits $v_{i,0,j}$, $v_{i,1,j}$ as input to $\gamma^{(3)}$ than those encrypted in ciphertexts $\Gamma_{i,0,j}, \Gamma_{i,1,j}$. To cope with fully-malicious adversaries, we replace 2PC evaluation of $\gamma^{(3)}$ from Figure 3 by using ZK Protocol (1). More specifically, instead of 2PC evaluation of $\gamma^{(3)}$, we run ZK Protocol (1) for circuit $\gamma^{(3)}$ with both the $\iota_{1,i}$ and the $v_{i,0,j}, v_{i,1,j}$ as P_1's input bits, and the $\iota_{2,i}$ as P_2's input bits. To run ZK Protocol (1), P_1 sends encryptions $\Gamma_{i,0,j}, \Gamma_{i,1,j}$ to P_2 (as well as dummy encryptions of the $\iota_{1,i}$). As a result of running ZK Protocol (1) of $\gamma^{(3)}$ instead of direct 2PC of $\gamma^{(3)}$, P_2 can verify that the $\Gamma_{i,0}, \Gamma_{i,1}$ are correct encryptions of P_1's input to $\gamma^{(3)}$. Note that the output bits received by P_2 after running ZK Protocol (1) comprise all output bits of circuit $\gamma^{(3)}$.

Also, P_1 could cheat by sending pairs $\Gamma_{i,0,0}$ and $\Gamma_{i,1,0}$ which are not encrypting 0 and 1. Hence, when P_1 sends the $\Gamma_{i,0,j}, \Gamma_{i,1,j}$ to P_2,



P_1
(input $\iota_{1,1}, \ldots, \iota_{1,\ell_1}$)
P_2
(input $\iota_{2,1}, \ldots, \iota_{2,\ell_2}$)
$\forall i \in \{1, \ldots, n\}$:
$\Gamma_{i,0,0} \leftarrow \mathsf{Enc}(0)$
$\Gamma_{i,1,0} \leftarrow \mathsf{Enc}(1)$
$\forall j \in \{1, \ldots, \lambda\}$:
$[v_{i,0,j}, v_{i,1,j} \overset{\$}{\leftarrow} \{0,1\}^2$
$\Gamma_{i,0,j} \leftarrow \mathsf{Enc}(v_{i,0,j})$
$\Gamma_{i,1,j} \leftarrow \mathsf{Enc}(v_{i,1,j})]$
$\forall j \in \{0, \ldots, \lambda\}$: $\Gamma_{i,0,j}, \Gamma_{i,1,j}$
2PC of $\gamma^{(3)}$ (see text)
$\forall i \in \{1, \ldots, n\}$:
$\Gamma_{i,2,0} \leftarrow \mathsf{Enc}(\omega_i)$
$\forall j \in \{1, \ldots, \lambda\}$:
$[\Gamma_{i,2,j} \leftarrow \mathsf{Enc}(v_{i,\omega_i,j})]$
$\forall j \in \{0, \ldots, \lambda\}$ $\Gamma_{i,2,j}$
$\Gamma_{i,0} = \sum_{j=0}^{\lambda} (2^{\lambda-j} \cdot \Gamma_{i,0,j})$
$\Gamma_{i,1} = \sum_{j=0}^{\lambda} (2^{\lambda-j} \cdot \Gamma_{i,1,j})$
$\Gamma_{i,2} = \sum_{j=0}^{\lambda} (2^{\lambda-j} \cdot \Gamma_{i,2,j})$
$\Delta_{i,0} = \Gamma_{i,0} - \Gamma_{i,2}$
$\Delta_{i,1} = \Gamma_{i,1} - \Gamma_{i,2}$
$\Gamma_{i,0} = \sum_{j=0}^{\lambda} (2^{\lambda-j} \cdot \Gamma_{i,0,j})$
$\Gamma_{i,1} = \sum_{j=0}^{\lambda} (2^{\lambda-j} \cdot \Gamma_{i,1,j})$
$\Gamma_{i,2} = \sum_{j=0}^{\lambda} (2^{\lambda-j} \cdot \Gamma_{i,2,j})$
$\Delta_{i,0} = \Gamma_{i,0} - \Gamma_{i,2}$
$\Delta_{i,1} = \Gamma_{i,1} - \Gamma_{i,2}$
$a_i \overset{\$}{\leftarrow} \mathbb{Z}_p, \pi \overset{\$}{\leftarrow} \Pi$
$\Delta'_{i,0} = a_i \cdot \Delta_{i,0}, \Delta'_{i,1} = a_i \cdot \Delta_{i,1}$
$\Delta'_{i,0}, \Delta'_{i,1}, \Delta'_{i,\pi(0)}, \Delta'_{i,\pi(1)}$
ZK proof Scalar$_i$
ZK proof Shuffle$_i$
if ZK proofs do not verify **then** abort
jointly decrypt $\Delta'_{i,\pi(0)}, \Delta'_{i,\pi(1)}$
if none or both decrypt to 0 **then** abort

Figure 3: ZK Protocol (3)

we require P_1 to additionally open $\Gamma_{i,0,0}$ and $\Gamma_{i,1,0}$, e.g., by sending random coins. Therewith, P_2 can verify that $\Gamma_{i,0,0}$ and $\Gamma_{i,1,0}$ are valid encryptions of 0 and 1.

3.3 Composition of ZK Protocols

Our ZK Protocols can be composed in a natural way, i.e., ZK Protocol (1), (2), and (3) can be jointly used on a single circuit γ. Protocol steps before and after 2PC evaluation of the modified circuit γ are simply executed in parallel. Different modifications of ZK Protocols (1) to (3) to circuit γ are merged into one large garbled circuit. This large circuit comprises γ's and all modifications' functionality and uses P_1's and P_2's input sets once. That is, inputs $\iota_{1,i}$ and $\iota_{2,i}$ are only used once and their wires are connected to all sub-functions of the large circuit. Obviously, all other necessary inputs $\mu_{i,j}$, $\sigma_{i,j}$, and $v_{\omega,j}$ are present for their respective input and outputs. This ensures

Input to $\gamma^{(3)}$

P_1	P_2
$\iota_{1,1}, \ldots, \iota_{1,\ell_1}$ $1 \le i \le n$: $[v_{i,0,1}, \ldots, v_{i,0,\lambda}, v_{i,1,1}, \ldots, v_{i,1,\lambda}]$	$\iota_{2,1}, \ldots, \iota_{2,\ell_2}$

Output of $\gamma^{(3)}$

```
1 $(\omega_1, \ldots, \omega_n) = \gamma((\iota_{1,1}, \ldots, \iota_{1,\ell_1}), (\iota_{2,1}, \ldots, \iota_{2,\ell_2}))$;
2 for $i = 1$ to $n$ do
3     output $\omega_i || v_{i,\omega_i,1} \cdots v_{i,\omega_i,\lambda}$;
```

Figure 4: Definition of circuit $\gamma^{(3)}$

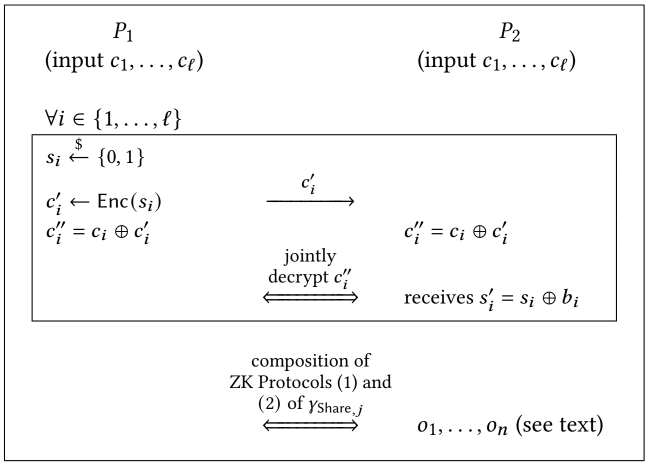

P_1
(input $c_1, \ldots, c_\ell$)
P_2
(input $c_1, \ldots, c_\ell$)
$\forall i \in \{1, \ldots, \ell\}$
$s_i \xleftarrow{\$} \{0,1\}$
$c'_i \leftarrow \mathsf{Enc}(s_i)$
c'_i
$c''_i = c_i \oplus c'_i$
$c''_i = c_i \oplus c'_i$
jointly decrypt c''_i
receives $s'_i = s_i \oplus b_i$
composition of ZK Protocols (1) and (2) of $\gamma_{\mathsf{Share},j}$
$o_1, \ldots, o_n$ (see text)

Figure 5: Conversion from FHE to 2PC for C_j^{Bool}

the same functionality of the large circuit as the sub-functions due to its security against malicious adversaries. Protocol steps outside of 2PC operate on distinct inputs and hence are non-interfering under parallel composition.

3.4 Conversion

3.4.1 FHE to 2PC. Let ciphertexts c_i be homomorphic encryptions of bits b_i. Parties P_1 and P_2 know the c_i, but not the b_i and want to evaluate Boolean circuit $C_j^{\mathsf{Bool}}(b_1, \ldots, b_\ell)$.

Figure 5 presents conversion details. As part of the joint decryption of c''_i, where only P_2 learns s'_i, P_1 must proof that its participation to decryption is correct [2, 5, 8].

Applying ZK Protocols (1) and (2): For each ciphertext c_i, P_1 knows s_i, and P_2 knows s'_i with $s_i \oplus s'_i = b_i$. Thus, parties have secret shared each b_i. Ciphertexts c'_i are encryptions of P_1's share, and c''_i are P_2's share.

To compute output bits $(o_1, \ldots, o_n) = C_j^{\mathsf{Bool}}(b_1, \ldots, b_\ell)$, both parties agree to evaluate Boolean circuit $\gamma_{\mathsf{Share},j}$, defined as

$$\begin{aligned}\gamma_{\mathsf{Share},j}((s_1, \ldots, s_\ell), (s'_1, \ldots, s'_\ell)) &= C_j^{\mathsf{Bool}}(s_1 \oplus s'_1, \ldots, s_\ell \oplus s'_\ell) \\ &= (o_1, \ldots, o_n).\end{aligned}$$

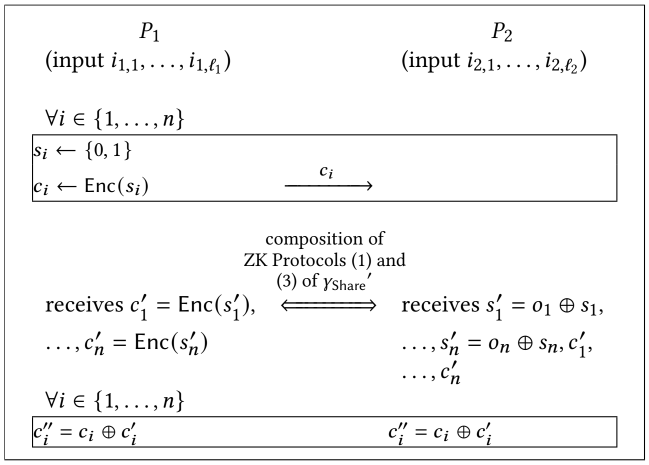

P_1
(input $i_{1,1}, \ldots, i_{1,\ell_1}$)
P_2
(input $i_{2,1}, \ldots, i_{2,\ell_2}$)
$\forall i \in \{1, \ldots, n\}$
$s_i \leftarrow \{0,1\}$
$c_i \leftarrow \mathsf{Enc}(s_i)$
c_i
composition of ZK Protocols (1) and (3) of γ_{Share}'
receives $c'_1 = \mathsf{Enc}(s'_1)$, $\ldots, c'_n = \mathsf{Enc}(s'_n)$
receives $s'_1 = o_1 \oplus s_1$, $\ldots, s'_n = o_n \oplus s_n, c'_1$, $\ldots, c'_n$
$\forall i \in \{1, \ldots, n\}$
$c''_i = c_i \oplus c'_i$
$c''_i = c_i \oplus c'_i$

Figure 6: Conversion from 2PC to FHE for C_j^{Bool}

Observe that $\gamma_{\mathsf{Share},j}$ has exactly the same input-output structure as γ. To make sure that P_1 uses inputs s_i matching homomorphic encryption c_i in γ_{Share}, we apply ZK proof (1). The proof uses circuit $\gamma_{\mathsf{Share},j}$ with inputs $(s_1, \ldots, s_\ell)$, $(s'_i, \ldots, s'_\ell)$ and ciphertexts $(c_1, \ldots, c_\ell)$. Also, P_2 must prove that they use s'_i, matching c''_i during the 2PC part of ZK proof (1). Therefore, we run a composition of ZK proof (1) and ZK proof (2) on circuit $\gamma_{\mathsf{Share},j}$.

Output bits: Output bits o_i of $\gamma_{\mathsf{Share},j}$ are the same bits output by C_j^{Bool}. However, if C_j^{Bool} is not the last sub-circuit of F, but another arithmetic circuit C_{j+1}^{Arith} is following, $\gamma_{\mathsf{Share},j}$ cannot output the o_i in the clear. Instead, we have to transform output bits into FHE encryptions which is part of the following 2PC to FHE conversion.

3.4.2 2PC to FHE. Parties want to evaluate $(o_1, \ldots, o_n) = C_j^{\mathsf{Bool}}((\iota_{1,1}, \ldots, \iota_{1,\ell_1}), (\iota_{2,1}, \ldots, \iota_{2,\ell_2}))$ such that in the end only FHE ciphertexts $\mathsf{Enc}(o_i)$ are known to both parties. Observe that bits $\iota_{1,i}$ and $\iota_{2,i}$ are either plain inputs bits from P_1 and P_2 or shares from the above FHE to 2PC conversion.

Figure 6 depicts conversion details. After P_1 has sent ciphertexts c_i of shares s_i of output bits o_i to P_2, parties evaluate circuit γ_{Share}', defined as

$$\begin{aligned}&\gamma_{\mathsf{Share}}'((\iota_{1,1}, \ldots, \iota_{1,\ell}, s_1, \ldots, s_n), (\iota_{2,1}, \ldots, \iota_{2,\ell})) \\ &\quad = (s_1, \ldots, s_n) \oplus C_i^{\mathsf{Bool}}((\iota_{1,1}, \ldots, \iota_{1,\ell}), (\iota_{2,1}, \ldots, \iota_{2,\ell})) \\ &\quad = (s_1 \oplus o_1, \ldots, s_n \oplus o_n) = (s'_1, \ldots, s'_n),\end{aligned}$$

such that P_2 receives output s'_i. Therewith, parties know their shares s'_i and s'_i of output bits o_i. Party P_2 knows c_i encrypting P_1's share.

There is another subtlety regarding the proof of correctness of all s_i and c_i. Party P_1 could again run ZK Protocol (1) for γ_{Share}'. This is, however, not enough. Circuit γ_{Share}' outputs s'_i to P_2, so P_2 must additionally compute $c'_i \leftarrow \mathsf{Enc}(s'_i)$, send the c'_i to P_1, and prove correctness. To achieve both, parties compose ZK Protocol (1) and ZK Protocol (3) on circuit γ_{Share}'. Note that P_2 computes the c'_i as part of ZK Protocol (3).

Finally, both parties compute all n encryptions $c''_i = \mathsf{Enc}(o_i) = \mathsf{Enc}(s_i \oplus s'_i)$ using c_i and c'_i and the homomorphic property of Enc to continue evaluation.

3.5 Supplementary Security Techniques

3.5.1 $d \geq 2$ parties. Secure multi-party computation can be constructed from secure two-party computations in various ways. One way is a star topology as we will use in our example in Section 4. The main idea is that each party P_i engages in secure two-party computation with a central party P_1 to compute some functionality. Such a centralized approach works for certain functionalities, e.g., equality of inputs, as equality is symmetric and transitive. If P_i's input is equal to P_1's and P_j's input is equal to P_1's, then P_i's input is also equal to P_j's. Hence, computation of the joint result using homomorphic encryption can leverage this relation.

However, this approach does not apply to other functionalities, e.g., larger-than comparison. If P_i's input is larger than P_1's and P_j's input is larger than P_1's, then we cannot imply any larger-than relation between P_i's and P_j's input. Consequently in this case, the alternative to maintain constant-round complexity is to engage all parties in pair-wise comparisons. This has been previously considered, e.g., in the context of sealed-bid auctions [7]. However, the result of each pairwise comparison is leaked in previous work, reducing security to a level comparable with order-preserving encryption. In contrast, constructions in this paper enable computing the auction result, e.g., the largest input, using homomorphic encryption with constant round complexity.

In summary, there exist several practically relevant protocols with arithmetic relations between inputs which can be decomposed into an initial two-party phase followed by a combination phase of the inputs. Maliciously-secure two-party protocols of the first phase can be efficiently implemented in a constant number of rounds. For low multiplicative-depth combination phases, we can use homomorphic encryption efficiently, too.

3.5.2 Reducing communication complexity. The costliest operation during garbled circuit 2PC evaluation in ZK Protocols (1) and (2) is verification of commitments $\mathsf{Com}_{i,j}$. If commitments are hash-based, then $\gamma^{(1)}$ and $\gamma^{(2)}$ would need to comprise sub-circuits recomputing, e.g., expensive SHA2 hashes.

However with a white-box use of garbled circuits, verifying commitments is unnecessary. Consider, first, ZK Protocol (1): instead of re-computing commitments in $\gamma^{(1)}$, evaluator P_2 simply retrieves wire labels $L_{i,j}$ of their input wire $\sigma_{i,j}$ from garbler P_1. During evaluation of $\gamma^{(1)}$, P_1 does not send the standard "translation-table" which opens the label of output wire $t_{i,j}$ by mapping the label to a 0 or 1. Instead, P_1 only sends a commitment to the table. After 2PC evaluation, P_2 sends label $L_{i,j}$, $\sigma_{i,j}$, and $R_{i,j}$ to P_1, P_1 verifies $\mathsf{Com}_{i,j}$, checks whether $L_{i,j}$ is the right label, and then sends the translation table.

In case of ZK Protocol (2) the situation is more subtle. P_1 needs to reveal both wire labels for $\sigma_{i,j} = 0$ and $\sigma_{i,j} = 1$ in order to prove integrity of its input. However, P_1 can only do so after P_2 has revealed output $t_{i,j}$, but before P_2 has opened the ciphertexts. Hence, another half communication round is necessary where the P_2 sends $t_{i,j}$ after evaluating the protocol. This order of operations is similar to the zero-knowledge proof technique using garbled circuits by Jawurek et al. [29] where the garbler opens the circuit after a commitment to the output by the evaluator. Note that our protocols secure the garbled circuit computation (in combination with conversion from and to FHE) whereas Jawurek et al. only construct a single ZKP using garbled circuits.

3.5.3 Supporting larger plaintext spaces. Our presentation above describes arithmetic sub-circuits C_i^{Arith} operating over single bits. That is, each ciphertext encrypts a single bit and homomorphic operations are over bits. This can be inefficient as parties often want to compute on larger integers, e.g., 32 Bit integers. Homomorphic encryption schemes anyways operate over large plaintext spaces, where addition of a large, multiple bit integer is a single homomorphic operation. A large plaintext space also allows for SIMD techniques.

To improve performance, we can extend conversion from operating over $GF(2)$ plaintexts to operate over plaintexts of arbitrary fields $GF(q)$ by instituting the following two modifications. In our conversions, ZK Protocols, and ZK proofs, we replace using XORs to share a single bit or combine two shares to a bit by additions and subtractions over $GF(q)$. Random bits serving as a share for a party become random elements of $GF(q)$. Second, n single bit encryptions $c_i = \mathsf{Enc}(b_i)$ output by our 2PC to FHE conversion are combined to a single n bit encrypted integer by each party computing $\sum_{i=0}^{n-1} 2^i \cdot c_{i+1}$.

3.6 Security Analysis

Our ZK Protocols (1) to (3) prove that the plaintext of an FHE ciphertext (under a shared key) and the input or output, respectively, of a 2PC are identical. They hence enable to compose FHE computations with 2PC protocols in a joint protocol that is secure against malicious adversaries.

Theorem 3.1. *ZK Protocols (1) to (3) are (a) complete, i.e., an honest verifier accepts the proof, if the prover provides consistent input, (b) zero-knowledge, i.e., any verifier learns nothing about the prover's witness except that it satisfies the proof, and (c) sound, i.e., honest verifier rejects the proof with overwhelming probability in the security parameter λ, if the prover's secret input is not a witness for the proof.*

Proof. Completeness of ZK Protocols (1) to (3) follows immediately from their construction, so we focus on Zero-Knowledge and Soundness.

Zero-Knowledge: To prove zero-knowledge, we construct simulators in the hybrid model who do not know the witness of the individual ZK Protocols (ZKPs), create views for the adversary which are indistinguishable from the real protocol, and make the verifier accept the proofs. In the hybrid model, simulators can simulate any ZK sub-proofs invoked during the protocol.

First, observe that all messages from the prover to the verifier are semantically-secure ciphertexts, random numbers or other zero-knowledge proofs.

In ZKP (1) and (2), the simulator randomly chooses inputs $\iota_{1,i}$ (or $\iota_{2,i}$) and masking bits $\mu_{i,j}$ as their input into 2PC. The verifier inputs $\sigma_{i,j}$ to the 2PC. After the 2PC, the simulator either receives verification bits $t_{i,j}$ (ZKP (1)) or outputs random verification bits (ZKP (2)).

In the last step, we make use of the hybrid model. The simulator invokes the simulator of the ZKP for correct decryption using those (random) verification bits and the committed (random) input and masking ciphertexts, simulating a consistent execution of the ZKP.

In ZKP (3), the simulator does not have to output verification bits $v_{i,\omega_i,j}$, but the verification is done using ZK proofs Scalar_i and Shuffle_i. Hence, the simulator for ZK Protocol (3) chooses a random ω_i and invokes the simulators for Scalar_i and Shuffle_i.

Soundness: To prove soundness for ZKP (1) and (2), we construct extractors. We construct an extractor only for ZKP (1), but stress that the extractor for (2) is equivalent. The extractor starts the ZK proof and lets the prover commit to their inputs via homomorphic ciphertexts $c_{1,j}$ (for a known shared key). Then the extractor chooses challenge bits $\sigma_{i,j}$ and sends them to the 2PC. The prover outputs verification bits $t_{i,j}$. The extractor rewinds the prover to just before they received the challenge bits for the 2PC. The extractor negates all challenge bits to $\neg\sigma_{i,j}$, sends them to the 2PC and continues the protocol. Let the prover's verification bits after rewinding be $t'_{i,j}$. We assume that the prover has consistent inputs and hence these inputs are extractable: the prover's inputs in ZKP (1) are $t_{i,j} \oplus t'_{i,j}$.

The soundness of ZKP (3) is a special case of authenticity of garbled circuits [4]. The challenge bits, $v_{i,0,j}$ and $v_{i,1,j}$, are input to the 2PC. Note that the soundness of the ZKP (1) ensures that the entire execution of the verifier is secure against malicious behaviour, including its conversion of the challenge bits from FHE to 2PC. The output depends on the output of the 2PC. Since the prover only evaluates the garbled circuit, it is bound to the correct or no output due to the authenticity property of garbled circuits. It can hence only produce one consistent set of output labels $v_{i,\omega_i,j}$.

This completes our security proof. Note that only the proof of ZKP (3) is recursive to the proof of ZKP (1), and hence all proofs are valid if ordered from (1) to (3). □

4 APPLICATION TO PRIVATE SET DISJOINTNESS

To indicate their usefulness, we apply our mixed-technique conversions to the area of private set analytics. In particular, we design a new solution to the problem of securely, yet efficiently computing private set disjointness (PSD). In PSD, parties compute whether their sets' intersection is empty. While protocols computing PSD have been presented before [17, 20, 27, 32, 33, 42, 61], our new solution features several advantages which, in combination, is unique: any number of $d \geq 2$ parties, fully-malicious security, circuit-based computations, and high efficiency (also due to a constant number of rounds). Computing PSD with a circuit-based approach is of special interest, as variations of PSD, like whether the size of the intersection is larger than a threshold, or other set statistics can then be computed easily, see discussions in [50, 52].

Let there be d parties $P_1, \ldots, P_d$. Each P_i has an n element input set $S_i = \{e_{i,1}, \ldots, e_{i,n}\}$, where for each element $e_{i,j}$ we have $e_{i,j} \in \{0,1\}^\ell$. We present a protocol where parties securely compute whether the intersection of the S_i is empty, i.e., $|\bigcap_{i=1}^d S_i| \overset{?}{=} 0$. Crucially, we do not leak the size of the intersection or any other information about the intersection or elements $e_{i,j}$.

Assume that parties have previously computed a distributed private key with corresponding public key for a fully or somewhat homomorphic encryption scheme. Separately, each party P_i has a public-private key pair, where the public key is known to all parties. Therewith, parties can securely communicate.

4.1 PSD Protocol Overview

We present a new circuit-based approach to compute PSD. At its core, parties compare their elements by evaluating a Boolean subcircuit with pairwise 2PC in a star topology. The outcome of 2PC comparisons then serves as input to FHE evaluations.

Hash Table Preparation: Initially, parties hash their input elements into hash tables. This is a typical approach of recent protocols for PSI, see Pinkas et al. [51] for an overview. Specifically, each party P_i starts by creating an empty hash table T_i with $m \in O(\frac{n}{\log n})$ buckets. To cope with possible hash collisions with very high probability, each bucket comprises a total of $\beta \in O(\log n)$ entries [53, 55]. Each entry has space to store ℓ bits. Let $T_i[j,k]$ denote the kth entry in the jth bucket $T_i[j]$ of P_i's hash table T_i.

After initializing hash table T_i, each party P_i iterates over their input elements, writing element $e_{i,j}$ into bucket $T_i[h(e_{i,j}), u]$, where u is the first empty entry in T_i's mth bucket. All remaining entries in the hash table are filled with random bit strings.

Mixed-Circuit Evaluation: Parties elect a leader, w.l.o.g. the leader is P_1. The main idea to compute PSD is that, for a randomly chosen r, the following function F is evaluated securely:

$$F = r \cdot \sum_{j=1}^{m} \sum_{k=1}^{\beta} \prod_{i=2}^{d} \left[\bigvee_{u=1}^{\beta} (T_1[j,k] \overset{?}{=} T_i[j,u]) \right].$$

Function F implements PSD, as sets S_i are disjoint *iff* F evaluates to 0. The rationale behind F is that the intersection is not empty if and only if there is exists an entry in a bucket of P_1's table which equals an entry of the same bucket in all other parties' tables.

We already define F using a mixed arithmetic and Boolean notation, suggesting a direct application of our maliciously secure mixed-techniques for 2PC-FHE evaluation.

To securely evaluate F, we set up a simple star topology where leader P_1 interacts pairwise with each other party P_i to compute inner parts $f_{i,j,k} = \left[\bigvee_{u=1}^{\beta} (T_1[j,k] \overset{?}{=} T_i[j,u]) \right]$ with 2PC. That is, for the kth entry in their jth bucket $T_1[j,k]$, P_1 evaluates with P_i a separate 2PC circuit which implements $f_{i,j,k}$. Using our 2PC to FHE conversion, output of each $f_{i,j,k}$ 2PC evaluation is a homomorphic encryption of its output bit which we denote by $\mathsf{Enc}(f_{i,j,k})$. After all 2PC computations, P_1 sends the $\mathsf{Enc}(f_{i,j,k})$ to all other parties which continue computing F homomorphically.

The final multiplication of the output by (a random) r in the encrypted domain is realized by each party P_i randomly selecting $r_i \overset{\$}{\leftarrow} M$ and sending $\mathsf{Enc}(r_i)$ to other parties. All parties homomorphically compute $\mathsf{Enc}(r) = \sum_{i=1}^{d} \mathsf{Enc}(r_i)$ and multiply the output by $\mathsf{Enc}(r)$ to get $\mathsf{Enc}(F)$. This ciphertext $\mathsf{Enc}(F)$ is then jointly decrypted. Without multiplying by r, parties would learn the size of the intersection.

Although the 2PC protocol, our conversion, and homomorphic evaluations are secure against malicious adversaries, we need to extend our current security model from two parties to the case of d parties. We do this in the next section.

4.2 Malicious Security for PSD

So far, our conversion is secure only for the case of $d = 2$ parties, where at most one party is malicious. Recall that after 2PC to FHE conversion of Figure 6, both parties P_1 and P_i have proven to each other correct computation of $c = \mathsf{Enc}(s)$ and $c' = \mathsf{Enc}(s')$. They homomorphically combine c and c' to $\mathsf{Enc}(f_{i,j,k}) = \mathsf{Enc}(s \oplus s')$. The new challenge when dealing with $d > 2$ parties is that both P_1 *and* P_i can be malicious, fabricate various different $\mathsf{Enc}(f_{i,j,k})$, and send different $\mathsf{Enc}(f_{i,j,k})$ to different other parties.

To mitigate, one could somehow run ZK proofs in public such that all other parties automatically observe the correct $\mathsf{Enc}(f_{i,j,k})$, but this is expensive. A more elegant solution would be that both parties P_1 and P_i sign $\mathsf{Enc}(f_{i,j,k})$ at the end of their conversion, and P_i sends their signature to P_1. Then, P_1 could use a secure echo broadcast [23] to send $\mathsf{Enc}(f_{i,j,k})$ and both signatures of $\mathsf{Enc}(f_{i,j,k})$ to all parties. As a result, all parties would receive the same $\mathsf{Enc}(f_{i,j,k})$ and verify that P_1 and P_i have agreed on it.

P_1 and P_i malicious: However, an interesting new challenge occurs when both P_1 and P_i are malicious and agree on a wrong $\mathsf{Enc}(f_{i,j,k})$. For example, P_1 and P_i could agree on $\mathsf{Enc}(0)$ even though P_i has an entry $e_{i,u}$ in its j^{th} bucket which equals an entry $e_{1,k}$ in P_1's j^{th} bucket. Note that this is not an attack, as the adversary can anyways control P_i's input and set it to arbitrary values. So, the above case would be equivalent to the adversary setting P_i's input $e_{i,u}$ to something different from $e_{1,k}$ in the first place. The only property P_1 and P_i have to prove to all other parties is that ciphertext $\mathsf{Enc}(f_{i,j,k})$ encrypts a bit.

As neither P_1 nor P_i know $f_{i,j,k}$, we use to a different strategy. Party P_1 proves in ZK that c encrypts a bit, and P_i proves that c' encrypts a bit. Parties broadcast c and c' with both proofs. Using c and c' all parties compute $\mathsf{Enc}(f_{i,j,k})$ homomorphically.

Finally, to force P_1 to always use the same inputs during pairwise comparisons with different P_i, we require P_1 to initially commit to its input using FHE ciphertexts and securely broadcast those ciphertexts to all other parties. The consistency of inputs can then be verified using ZK Protocol (1).

Joint decryption: Recall that the 2PC to FHE conversion internally runs ZK Protocol (3) and requires a joint decryption between P_1 and P_i. In the case of $d > 2$ parties, joint decryption is still possible, but involves all d parties. So, both P_1 and P_i broadcast a request to decrypt the current $\Delta'_{i,\pi(0)}$ and $\Delta'_{i,\pi(1)}$, and all parties reply to P_1 with their share of the decryption (plus proof of correct decryption). Note that this does not change our total message complexity. We need to run $O(1)$ broadcasts for each $f_{i,j,k}$ anyways.

4.3 Implementation

We have implemented our private set disjointness variant with 2PC to FHE conversion and performed micro-benchmarks in a security setting we dub "semi-malicious". While our implementation of 2PC-part $f_{i,j,k}$ in the framework by Wang et al. [60] is maliciously secure, none of the common FHE libraries (HELib, PALISADE, SEAL, TFHE) provides all features we need for maliciously-secure conversion. Moreover, the implementation of a FHE scheme with threshold decryption and ZK proofs, e.g., based on the one by Asharov et al. [2], deserves its own paper. Thus, for the arithmetic part of F, we have only implemented and benchmarked arithmetic operations with FHE, but not FHE ZK proofs.

Table 1: Online time (s) for evaluating F, our scheme vs. MP-SPDZ vs. MP-BMR vs FHE. 2PC: communication time for circuit evaluation of all $m\beta d$ circuits $((\gamma_{\mathsf{Share}}'(1))(3))(1)$, BC: communication time for broadcasting shares and partial decryptions, FHE Comp: computation time for arithmetic part, DNF: does not finish in 15min

		Ours						
n	d	2PC	BC	FHE Comp	**Total**	**SPDZ Total**	**BMR Total**	**FHE Total**
32	5	2.2	1.1	1.0	**4.3**	**16.4**	**8.5**	**141.7**
	10	3.9	1.8	1.8	**7.5**	**33.1**	**24.3**	**283.0**
	20	7.6	5.5	3.6	**16.6**	**50.3**	**Crash**	**565.5**
	40	14.8	17.6	7.1	**39.5**	**215.7**	**Crash**	**DNF**
64	5	4.7	1.4	2.3	**8.4**	**35.6**	**18.5**	**406.9**
	10	9.0	3.4	4.4	**16.8**	**72.4**	**66.6**	**813.1**
	20	18.0	10.7	8.6	**37.3**	**248.2**	**Crash**	**DNF**
	40	35.9	40.9	17.0	**93.8**	**784.3**	**Crash**	**DNF**
128	5	10.7	2.2	5.4	**18.3**	**117.5**	**43.0**	**DNF**
	10	20.8	6.6	10.3	**37.7**	**356.7**	**Crash**	**DNF**
	20	41.8	24.2	20.1	**86.1**	**675.8**	**Crash**	**DNF**
	40	83.3	95.3	39.7	**218.3**	**DNF**	**Crash**	**DNF**
1024	5	121.2	17.5	61.6	**200.4**	**DNF**	**DNF**	**DNF**
2048	5	265.0	37.5	135.52	**438.0**	**DNF**	**DNF**	**DNF**

More specifically, we have implemented the actual circuit which is evaluated as part of the 2PC to FHE conversion of $f_{i,j,k}$, namely $((\gamma_{\mathsf{Share}}'(1))(3))(1)$. Here, circuit γ_{Share}' is the modification to $f_{i,j,k}$ due to conversion, $\gamma_{\mathsf{Share}}'(1)$ is the modification implied by ZK Protocol (1) on top of that, $(\gamma_{\mathsf{Share}}'(1))(3)$ the modification by ZK Protocol (3) on top of that, and $((\gamma_{\mathsf{Share}}'(1))(3))(1)$ the modification by ZK Protocol (1) running inside ZK Protocol (3).

While all FHE frameworks available today allow straightforward specification of arithmetic operations, the framework we chose for implementing the arithmetic part of F was TFHE [12]. Other libraries require significant preparatory work to compute performance-optimal parameters, but TFHE can run off-the-shelf with the parameters from Chillotti et al. [13].

For all benchmarks, we set $m = \frac{n}{2}$, $\beta = \log n$, and consider $\ell = 32$ bit integers as the elements in each party's set. It is well known that communication time due to latency between parties is a dominating factor regarding total runtime, especially for the 2PC part. For example, raw computation time of evaluating a single $((\gamma_{\mathsf{Share}}'(1))(3))(1)$ circuit for $\beta = 5$ takes only ≈ 1.2 ms on a single 1.6 GHz Core i5, but all computations can run in parallel on different cores. So, an Amazon EC2 C5d instance with 96 cores computes $\approx 80,000$ circuits per second. However, network traffic, i.e., exchanging the ≈ 177 KByte of data between P_1 and P_i during evaluation of that circuit, cannot be parallelized. Instead, we can only sequentially sent all data for all circuits, and network latency is here the crucial parameter. While latency of (intercontinental) WAN traffic is often unstable and can go over 250 ms [59], we run benchmarks on one machine to better control network behavior and use `netem` [47] to set latency to a modest 70 ms. As a result of this latency, we measured data goodput over TCP to be only

≈ 330 MBit/s on the `localhost` network (a higher latency would imply less goodput).

Following the 2PC phase, we have benchmarked the remaining arithmetic operations, performed separately by each party, with TFHE. Computation time to evaluate any bootstrapped gate was ≈ 105 ms on a single core, and again each of the arithmetic operations (XORing shares, multiplying for all d, the two sums) can be parallelized. On an Amazon C5d instance with 96 cores, roughly 914 gates can be evaluated per second.

Table 1 presents timings ignoring one time costs for preprocessing and initial, once and for all data exchange (only online time). To put our timings in perspective, we have also implemented and benchmarked F in the SPDZ implementation and BMR-SPDZ implementation from MP-SPDZ [16, 31]. While there are several MPC frameworks readily available [24], we have chosen MP-SPDZ due to its simplicity for implementing new functionalities. Additionally, we have also benchmarked the time it takes to compute F when this is implemented in only semi-honest FHE (with TFHE). There, circuit $f_{i,j,k}$ is also implemented in FHE.

In Table 1, 2PC denotes the time to compute all $((\gamma_{\mathsf{Share}}{}'(1))(3))(1)$. BC denotes the time for all broadcasts, i.e., the time broadcast all shares c_i, c_i' after 2PC to all parties (one TFHE ciphertext has size 2.5 KByte) plus the time to broadcast a partial decryption of the final result after FHE from each party (we assume that a partial decryption is one TFHE ciphertext). FHE Comp is the time, for each party, to compute the arithmetic part of F in TFHE. SPDZ Total and BMR Total are the total (online) time of MP-SPDZ to compute F with SPDZ or BMR. FHE Total is the total time of the semi-honest "pure-FHE" implementation, including broadcasting each party's $m\beta\ell$ ciphertexts to all other parties. Note that BMR crashed even for a small number of parties, e.g., $n = 128, d = 10$, or quickly run out of memory (32 GByte) for $d \geq 20$ parties.

Looking at Table 1, our implementation outperforms SPDZ, BMR and FHE in all considered settings. While SPDZ and BMR are competitive for a small number of parties, BMR fails due to its memory consumption and our composition from 2PC clearly shows better scalability than SPDZ for larger numbers of parties.

4.4 Complexity Analysis

We present and compare complexities of our mixed-techniques approach for evaluating F with related schemes. As there is no dedicated protocol for multi-party maliciously-secure PSD, we compare complexities with those for evaluating F using general MPC techniques SPDZ [15], constant-round MPC [40], and (semi-honest) FHE. Table 2 shows results of "online" phases only (SPDZ, constant-round MPC, our techniques). We stress that in contrast to our more detailed explanations below, Table 2 presents only a summary, focussing on those costs which dominate computation and communication. For example for the FHE-based approach, we silently ignore the n FHE additions in the outer part of F, as $O(d\ell n \log n)$ FHE multiplications will dominate total computation time. As mentioned above, we set $m \in O(\frac{n}{\log n})$ and $\beta \in O(\log n)$. To implement secure broadcast, we use the standard echo broadcast [15, 23, 40, 41] which has message complexity $O(d^2)$.

(Semi-Honest) FHE: Let $C_{\mathsf{FHE}*}$ be the computational complexity for a FHE multiplication and $C_{\mathsf{FHE}+}$ the computational complexity

Table 2: Complexities for Multi-Party Maliciously-Secure PSD using different techniques. Table shows *only online* phases (if applicable). Table lists *only dominating* computation or communication costs, see text.
λ: statistical security parameter, κ: computational security parameter, $\mathcal{I}$: total number of comparisons ($\mathcal{I} = d\ell n \log n$), d: number of parties, n : elements per party, ℓ : input length, $C_{GF(2^{\ell+\lambda})*}$: comp. cost for $GF(2^{\ell+\lambda})$ multiplication, H: comp. cost for hash evaluation, $|\mathsf{SYM}|$: size of symmetric ciphertext of $GF(2^{\ell+\lambda})$ element, $|H|$: size of a hash, $|\mathsf{FHE}|$: size of a FHE ciphertext, BC_x: secure broadcast of x bit, $\mathcal{I}$: total number of bit comparisons ($\mathcal{I} = d\ell n \log n$).

For practical scenarios, we simplify: $O(n\ell) \cdot BC_\kappa \subseteq O(\mathcal{I}) \cdot BC_{|\mathsf{FHE}|}$, $\ell d^3 \in O(d\mathcal{I})$, $\frac{\lambda \mathcal{I}}{\log n} + dn\lambda^2 \in O(\mathcal{I})$, $O(I{+}nd\lambda{\cdot}(\ell{+}\lambda)){\cdot}H \subseteq O(\mathcal{I}){\cdot}C_{\mathsf{FHE}*}$, $O(n{\cdot}(\ell{\cdot}(\log n{+}\lambda){+}\lambda^2)){\cdot}|H| \subseteq O(\mathcal{I}) \cdot |\mathsf{FHE}|$, $O(nd(\lambda\ell + \lambda^2)) \cdot |\mathsf{FHE}| \subseteq O(\mathcal{I}) \cdot |\mathsf{FHE}|$, $O(\ell) \cdot BC_{|H|} \subseteq O(nd) \cdot BC_{|\mathsf{FHE}|}$.

	Comp. / party	Comm. / party	Rounds
FHE	$O(\mathcal{I}) \cdot C_{\mathsf{FHE}*}$	$O(\ell n) \cdot BC_{\lvert\mathsf{FHE}\rvert}$	$O(1)$
Constant Round MPC [40]	$O(\mathcal{I}) \cdot C_{\mathsf{FHE}*}$	$O(\mathcal{I}) \cdot BC_{\lvert\mathsf{FHE}\rvert}$	$O(1)$
SPDZ [15]	$O(d\mathcal{I}) \cdot C_{GF(2^{\ell+\lambda})*}$	$O(d\mathcal{I}) \cdot \lvert\mathsf{SYM}\rvert{+}O(n){\cdot} BC_{\lvert GF(2^{\ell+\lambda})\rvert}$	$O(\log d + \log\log n)$
This paper	$O(\mathcal{I}) \cdot C_{\mathsf{FHE}*}$	$O(\mathcal{I}){\cdot}\lvert\mathsf{FHE}\rvert{+}O(n){\cdot}BC_{\lvert\mathsf{FHE}\rvert}$	$O(1)$

for a FHE addition. A standard FHE implementation arithmetizes F's inner part $f_{i,j,k}$. There, two ℓ Bit elements are compared with $O(\ell)$ multiplications (implementing XNORs and ANDs), followed by $\log n$ multiplications to realize $\bigvee$. Finally, d multiplications are necessary for $\prod$. In total, FHE requires $O(d\ell n \log n) \cdot C_{\mathsf{FHE}*}$ homomorphic multiplications with a multiplicative circuit depth of $\log \ell + \log\log n + \log d + 1$. Even for reasonable values $\ell = 32, d = 10$, $n = 2^{20}$, the multiplicative depth is already 14 which leads to huge runtimes in practice [44]. Note that homomorphic additions also increase ciphertext noise. While noise increased by additions is roughly one order of magnitude less than with multiplications [58], and we do not count additions in our comparison, we stress that additive noise requires FHE parameter selection to result in even slower computations.

Communication complexity with FHE comprises securely broadcasting all $(m \cdot \beta) \in O(n)$ input elements encrypted bit by bit and partial decryptions for the final ℓ Bit output. Such a standard FHE evaluation of F leads to a constant round complexity.

Constant-Round MPC: An implementation based on recent constant-round MPC protocols [31, 40, 41] replaces F's arithmetic operators with Boolean operators, i.e., the $\prod$ by $\bigwedge$ and each $\sum$ by $\bigvee$. The result is a circuit with $dn\ell$ input wires, $n\ell$ per party, one output wire, and $d\ell n \log n$ gates. This circuit is then evaluated in an online phase having the following complexities: (I) For each input wire of each party P_i, P_i broadcasts one PRG seed of length κ (security parameter), and all parties perform a distributed decryption, also broadcasting partial decryptions. (II) For each gate, all parties perform a distributed decryption. Together, per party, this requires a total of $O(d\ell n \log n)$ broadcasts of size comparable to a FHE ciphertext and $O(n\ell)$ broadcasts of PRG seeds. Lindell et al. [40] require 9 rounds and a FHE multiplicative depth of 3.

SPDZ: Comparing two ℓ Bit integers is implemented in SPDZ [15] by Catrina and de Hoogh [11]'s arithmetization. For statistical security parameter λ, each comparison requires $d \cdot \ell$ multiplications in $GF(2^{\ell+\lambda})$ per party, in a constant number of rounds. The following $\bigvee$ requires $\log n$ and the $\prod$ requires d multiplications. Opening the final output requires $O(\ell \cdot d^3)$ multiplications per party. So in total, F's evaluation requires $O(nd \log nd\ell + \ell d^3) = O(d^2 \ell n \log n + \ell d^3)$ multiplications per party in $O(\log d + \log \log n)$ rounds. This is also the amount of shares which have to be securely exchanged between two parties. Initial sharing of $O(n)$ elements of each party requires $O(n)$ secure broadcasts.

Our Mixed-Technique (FHE): Let $C_{2\mathsf{PC}}$ denote the computational complexity for computing the 2PC sub-protocol for inner circuits $f_{i,j,k}$ of F. For P_1, computational complexity for evaluating F is $O(nd) \cdot C_{2\mathsf{PC}}$ plus $O(nd) \cdot C_{\mathsf{FHE}*}$ plus $O(n) \cdot C_{\mathsf{FHE}+}$. So, the computational complexity is in $O(n \cdot (C_{\mathsf{FHE}+} + d \cdot (C_{2\mathsf{PC}} + C_{\mathsf{FHE}*})))$.

As the exact complexities of $C_{2\mathsf{PC}}$ require 2PC to FHE conversion and are relatively involving, we defer full details to Appendix A. In summary, $C_{2\mathsf{PC}}$ implies 2PC evaluation of a circuit of $O(\ell \cdot (\log n + \lambda) + \lambda^2)$ gates.

Regarding the FHE part of the conversion (details in Appendix A), we need $O(\lambda(\ell+\lambda))$ FHE multiplications $C_{\mathsf{FHE}*}$, $O(\lambda(\ell+\lambda))$ encryptions, one decryption, one ZK proof Scalar, and one ZK proof Shuffle per comparison. Evaluation of F's remaining (outer) arithmetic part adds dn FHE multiplications which, however, increase multiplicative circuit depth to $2+\log d$. Note that this is a significant improvement over the pure FHE approach above. For example, with $d = 10$ parties, multiplicative depth is only 6, independent of n.

We present communication complexity in Appendix A, too, and only summarize here. For comparisons $f_{i,j,k}$, we send $O(n \cdot (\ell \cdot (\log n + \lambda) + \lambda^2))$ hash values per party and $O(nd(\lambda\ell + \lambda^2))$ FHE ciphertexts for party P_1 for the conversion. In total, P_1 also broadcasts $O(nd)$ FHE ciphertexts and $O(\ell)$ commitments to their input.

The total number of rounds is asymptotically constant in d, ℓ, λ, and n, and it is low in practice: only 6 rounds are necessary for the online phase, see Appendix A for details. To construct authenticated garbled tables, efficient implementations [19] realize preprocessing in 3 rounds, resulting in a total of 9 rounds.

The comparison of the complexities of our protocol confirms our assessment. FHE is the most communication efficient approach, but the evaluation shows that its running times are not yet practical for deep circuits. SPDZ's [15] deployment is challenged by its communication complexity and constant-round MPC protocols [40] are complex and require large amounts of memory. Our protocol and composition technique strikes a balance of communication cost and computational efficiency.

5 RELATED WORK

Mixed-Techniques MPC: Several previous works combine different MPC techniques to mitigate individual techniques' drawbacks. Kolesnikov et al. are among the first to present a conversion between garbled circuits and (additively) homomorphic encryption in the two-party semi-honest model [34, 36]. Extending their conversion to also support fully-malicious adversaries is non-trivial: in Appendix D of [35], they present honest-verifier zero-knowledge proofs which render the protocol secure only if, e.g., at most one party is malicious. However, HVZK is insufficient, if proofs are part of a multi-party scenario with more than two parties where more than one party can be malicious.

A long line of research has focused on making mixed-techniques very practical and efficient. Henecka et al. [26] design practical tools for conversion between two-party garbled circuits and additively homomorphic encryption. Their conversion targets semi-honest adversaries and circuits for two parties. Demmler et al. [18] present a general two-party framework to convert between arithmetic sharing, Boolean sharing, and garbled circuits in the semi-honest model, and so do Riazi et al. [54]. Mohassel and Rindal [45] extend this line of work to three parties with malicious security. Again in the semi-honest model for two parties, Juvekar et al. [30] switch between garbled circuits and additively homomorphic encryption based on fast lattice-based cryptography, and Büscher et al. [10] switch between arithmetic and Boolean sharing. Rotaru and Wood [56] and Aly et al. [1] convert between MPC based on arithmetic secret sharing and garbled circuits with malicious security.

In conclusion, this paper fills a gap by offering a solution which converts between FHE and garbled circuits, supports any number of parties, and provides malicious security.

(Multi-Party) PSI and Disjointness: While seminar works in PSI are based on dedicated protocols [43], recent papers use a circuit-based approach (see Pinkas et al. [49] for an overview), culminating in solutions with asymptotically optimal communication complexity and practical constants [52]. In theory, such circuit-based approaches can be used to also compute disjointness, but they all focus on the two-party setting with semi-honest security.

Hazay and Venkitasubramaniam [25] present a maliciously-secure multi-party PSI protocol based on oblivious polynomial evaluation (OPE). Similar to previous ideas [20], OPE could then be combined with a maliciously-secure 2PC to compute disjointness. However, already computing the intersection is expensive with this approach, requiring $O(n^2)$ modular exponentiations. Kolesnikov et al. [37] present an efficient multi-party PSI protocol in the semi-honest model using only symmetric encryption. However, PSI protocols cannot be easily converted into PSI analytics protocols while maintaining efficiency [50, 52]. Other works have considered computing set disjointness, but these target semi-honest security and/or only two parties [17, 20, 27, 32, 33, 42, 61]

In conclusion, this paper presents the first multi-party PSI analytics protocol whose communication complexity scales only quadratically in the number of participants d. Furthermore, it is also secure in the malicious model.

REFERENCES

[1] A. Aly, E. Orsini, D. Rotaru, N.P. Smart, and T. Wood. Zaphod: Efficiently Combining LSSS and Garbled Circuits in SCALE. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, November 11-15, 2019*, pages 33–44. ACM, 2019.

[2] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 483–501, 2012.

[3] D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols (Extended Abstract). In *22nd Annual ACM Symposium on Theory of Computing, STOC, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513, 1990.

[4] M. Bellare, V.T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796, 2012.

[5] R. Bendlin and I. Damgård. Threshold Decryption and Zero-Knowledge Proofs for Lattice-Based Cryptosystems. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 201–218, 2010.

[6] F. Benhamouda, J. Camenisch, S. Krenn, V. Lyubashevsky, and G. Neven. Better Zero-Knowledge Proofs for Lattice Encryption and Their Application to Group Signatures. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 551–572, 2014.

[7] E.-O. Blass and F. Kerschbaum. Strain: A Secure Auction for Blockchains. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *Lecture Notes in Computer Science*, pages 87–110. Springer, 2018.

[8] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P.M.R. Rasmussen, and A. Sahai. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 565–596, 2018.

[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, ITCS 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.

[10] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider. HyCC: Compilation of Hybrid Protocols for Practical Secure Computation. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 847–861. ACM, 2018.

[11] O. Catrina and S. de Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 2010.

[12] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast Fully Homomorphic Encryption Library, 2016. https://tfhe.github.io/tfhe/.

[13] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptology*, 33(1):34–91, 2020.

[14] I. Damgård and A. López-Alt. Zero-Knowledge Proofs with Low Amortized Communication from Lattice Assumptions. In *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*, pages 38–56, 2012.

[15] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[16] data61. Versatile framework for multi-party computation , 2019. https://github.com/data61/MP-SPDZ.

[17] A. Davidson and C. Cid. An Efficient Toolkit for Computing Private Set Operations. In *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part II*, pages 261–278, 2017.

[18] D. Demmler, T. Schneider, and M. Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015.* The Internet Society, 2015.

[19] Efficient Multi-Party Computation Toolkit. EMP-ag2pc, 2019. https://github.com/emp-toolkit/emp-ag2pc.

[20] M.J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 1–19, 2004.

[21] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[22] S. Goldfeder. A Boolean Circuit for SHA-256 , 2019. http://stevengoldfeder.com/projects/circuits/sha2circuit.html.

[23] S. Goldwasser and Y. Lindell. Secure Multi-Party Computation without Agreement. *J. Cryptology*, 18(3):247–287, 2005.

[24] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic. SoK: General Purpose Compilers for Secure Multi-Party Computation. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1220–1237. IEEE, 2019.

[25] C. Hazay and M. Venkitasubramaniam. Scalable Multi-party Private Set-Intersection. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*, pages 175–203, 2017.

[26] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 451–462. ACM, 2010.

[27] S. Hohenberger and S.A. Weis. Honest-Verifier Private Disjointness Testing Without Random Oracles. In *Privacy Enhancing Technologies, 6th International Workshop, PET 2006, Cambridge, UK, June 28-30, 2006, Revised Selected Papers*, pages 277–294, 2006.

[28] M. Ishaq, A. Milanova, and V. Zikas. Efficient MPC via Program Analysis: A Framework for Efficient Optimal Mixing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1539–1556, 2019.

[29] M. Jawurek, F. Kerschbaum, and C. Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 955–966, 2013.

[30] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1651–1669. USENIX Association, 2018.

[31] M. Keller and A. Yanai. Efficient Maliciously Secure Multiparty Computation for RAM. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 91–124. Springer, 2018.

[32] A. Kiayias and A. Mitrofanova. Testing Disjointness of Private Datasets. In *Financial Cryptography and Data Security, 9th International Conference, FC 2005, Roseau, The Commonwealth of Dominica, February 28 - March 3, 2005, Revised Papers*, pages 109–124, 2005.

[33] L. Kissner and D. Xiaodong Song. Privacy-Preserving Set Operations. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 241–257, 2005.

[34] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *Cryptology and Network Security, 8th International Conference, CANS 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings*, volume 5888 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2009.

[35] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. From dust to dawn: Practically efficient two-party secure function evaluation protocols and their modular design. Cryptology ePrint Archive, Report 2010/079, 2010. https://eprint.iacr.org/2010/079.

[36] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. *Journal of Computer Security*, 21(2):283–315, 2013.

[37] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. Practical Multi-party Private Set Intersection from Symmetric-Key Techniques. In *2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30-November 03, 2017*, pages 1257–1272, 2017.

[38] V. Kolesnikov, J.B. Nielsen, M. Rosulek, N. Trieu, and R. Trifiletti. DUPLO: Unifying Cut-and-Choose for Garbled Circuits. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 3–20. ACM, 2017.

[39] Y. Lindell. Fast Cut-and-Choose-Based Protocols for Malicious and Covert Adversaries. *J. Cryptology*, 29(2):456–490, 2016.

[40] Y. Lindell, N.P. Smart, and E. Soria-Vazquez. More Efficient Constant-Round Multi-party Computation from BMR and SHE. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 554–581, 2016.

[41] Y. Lindell, B. Pinkas, N.P. Smart, and A. Yanai. Efficient Constant-Round Multi-party Computation Combining BMR and SPDZ. *J. Cryptology*, 32(3):1026–1069, 2019.

[42] L. Marconi, M. Conti, and R. Di Pietro. CED^2: Communication Efficient Disjointness Decision. In *Security and Privacy in Communication Networks - 6th*

Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 290–306. Springer, 2010.

[43] C.A. Meadows. A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*, pages 134–137. IEEE Computer Society, 1986.

[44] C. Aguilar Melchor, M.-O. Killijian, C. Lefebvre, and T. Ricosset. A Comparison of the Homomorphic Encryption Libraries HElib, SEAL and FV-NFLlib. In *Innovative Security Solutions for Information Technology and Communications - 11th International Conference, SecITC 2018, Bucharest, Romania, November 8-9, 2018, Revised Selected Papers*, volume 11359 of *Lecture Notes in Computer Science*, pages 425–442. Springer, 2018.

[45] P. Mohassel and P. Rindal. ABY^3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018.

[46] S. Myers, M. Sergi, and A. Shelat. Threshold Fully Homomorphic Encryption and Secure Computation. *IACR Cryptology ePrint Archive*, 2011:454, 2011. URL http://eprint.iacr.org/2011/454.

[47] Netem. netem, 2019. https://wiki.linuxfoundation.org/networking/netem.

[48] J.B. Nielsen and C. Orlandi. LEGO for Two-Party Secure Computation. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 368–386. Springer, 2009.

[49] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private Set Intersection Using Permutation-based Hashing. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 515–530. USENIX Association, 2015.

[50] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient Circuit-Based PSI via Cuckoo Hashing. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, pages 125–157, 2018.

[51] B. Pinkas, T. Schneider, and M. Zohner. Scalable Private Set Intersection Based on OT Extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[52] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient Circuit-Based PSI with Linear Communication. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153. Springer, 2019.

[53] M. Raab and A. Steger. "Balls into Bins" - A Simple and Tight Analysis. In *Randomization and Approximation Techniques in Computer Science, Second International Workshop, RANDOM'98, Barcelona, Spain, October 8-10, 1998, Proceedings*, pages 159–170, 1998.

[54] M.S. Riazi, C. Weinert, O. Tkachenko, E.M. Songhori, T. Schneider, and F. Koushanfar. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 707–721. ACM, 2018.

[55] P. Rindal and M. Rosulek. Malicious-Secure Private Set Intersection via Dual Execution. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1229–1242, 2017.

[56] D. Rotaru and T. Wood. MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security. In *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*, volume 11898 of *Lecture Notes in Computer Science*, pages 227–249. Springer, 2019.

[57] M. Strand. A Verifiable Shuffle for the GSW Cryptosystem. In *Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers*, pages 165–180, 2018.

[58] M. Varia, S. Yakoubov, and Y. Yang. HEtest: A Homomorphic Encryption Testing Framework. In *Financial Cryptography and Data Security - FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*, volume 8976 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2015.

[59] Verizon. IP Latency Statistics, 2020. https://enterprise.verizon.com/terms/latency/.

[60] X. Wang, S. Ranellucci, and J. Katz. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, page 21–37, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468.

[61] Q. Ye, H. Wang, J. Pieprzyk, and X.-M. Zhang. Efficient Disjointness Tests for Private Datasets. In *Information Security and Privacy*, pages 155–169, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-70500-0.

[62] M. Yung. From Mental Poker to Core Business: Why and How to Deploy Secure Computation Protocols? In *22nd ACM SIGSAC Conference on Computer and Communications Security, CCS'15, Denver, CO, USA, October 12-16, 2015*, pages 1–2, 2015.

A COMPLEXITY DETAILS

Table 3: Asymptotic circuit complexity. In our notation, "+x" for wires or gates means that $O(x)$ wires or gates are added by running a particular circuit.

	#Input wires (P_1, P_2)	#Output wires	#Gates
$f_{i,j,k}$	$\ell, \ell \log n$	1	$\ell \log n$
γ_{Share}'	$+1, +0$	$+0$	$+1$
$\gamma_{\mathsf{Share}}'(1)$	$+\lambda\ell, +\lambda\ell$	$+\lambda\ell$ (1 real)	$+\lambda\ell$
$(\gamma_{\mathsf{Share}}'(1))(3)$	$+\lambda, +0$	$+\lambda$ (1 real)	$+\lambda$
$((\gamma_{\mathsf{Share}}'(1))(3))(1)$	$+\lambda^2, +\lambda^2$	$+\lambda^2$ (1 real)	$+\lambda^2$
Total	$\ell \cdot (\log n + \lambda) + \lambda^2$	$\lambda\ell + \lambda^2$	$\ell \cdot (\log n + \lambda) + \lambda^2$

Circuit complexity: Comparison circuit $f_{i,j,k}$ has $O(\ell)$ input wires for P_1, $O(\ell \log n)$ for P_2, and one output wire (for P_2). Its number of gates is $O(\ell \log n)$, as two ℓ bit strings can be compared with $O(\ell)$ gates.

For our conversion from 2PC to FHE, we run γ_{Share}' of $f_{i,j,k}$, which adds additional complexity, see also Table 3. Specifically, running γ_{Share}' adds $O(1)$ input wires for P_1, no additional input wire to P_2, no additional output wire, and $O(1)$ additional gates (one XOR).

Running ZK Protocol (1) on γ_{Share}' leads to circuit $\gamma_{\mathsf{Share}}'(1)$. This circuit increases the number of input wires for P_1 by λ wires (the $\mu_{i,j}$) for each of P_1's ℓ input wires. It also increases P_2's input wires by $\lambda\ell$ input wires (choice bits $\sigma_{i,j}$). The number of output wires is increased by $\lambda\ell$ ($\mu_{i,j}$ or $\mu_{i,j} \oplus \iota_{i,j}$), and the number of gates, too (for-loop). Note that all but one output wire are used for ZK proofs, and one single wire carries the actual output from the previous circuit. ZK Protocol (3) is run, leading to $(\gamma_{\mathsf{Share}}'(1))(3)$. This circuit adds λ input wires for P_1 (the v), λ output wires (all but one used for ZK proofs), and λ gates. Finally, ZK Protocol (1) is run, resulting in $((\gamma_{\mathsf{Share}}'(1))(3))(1)$. This circuit adds $2\lambda^2$ input wires for P_1, i.e., λ wires ($\mu_{i,j}$) for each of the 2λ additional input wires from previous circuit $(\gamma_{\mathsf{Share}}'(1))(3)$. Input for P_2 is also increased by $2\lambda^2$ wires (λ wires for the $\sigma_{i,j}$ for each of P_1's additional input). Consequently, output wires are increased by $2\lambda^2$ ($\mu_{i,j}$ or $\mu_{i,j} \oplus \iota_{i,j}$), and the number of gates, too.

In total, our conversion leads to a circuit with $O(\ell \cdot (\log n + \lambda) + \lambda^2)$ input wires, $O(\lambda\ell + \lambda^2)$ output wires, and $O(\ell \cdot (\log n + \lambda) + \lambda^2)$ gates.

FHE complexity: 2PC to FHE conversion also involves additional FHE operations. That is, the part before and after $f_{i,j,k}$'s 2PC in Figure 6 requires 1 FHE encryption and 1 FHE multiplication. We then run ZK Protocol (1) which adds $\lambda\ell$ FHE encryptions and multiplications. Note that the depth of these multiplications is only 1. We then run ZK Protocol (3) which adds λ FHE encryptions, 1 ZK proof Scalar, 1 ZK proof Shuffle, and 1 decryption. Multiplicative depth remains 1. Finally, we run ZK Protocol (1) again, adding λ^2 FHE encryptions and multiplications, each of depth 1.

Number of rounds: Recall that the maliciously secure 2PC protocol by Wang et al. requires 3 rounds (steps 5 and 6 in Figure 2 in [60]) during online evaluation. The first two rounds comprise exchanging shares of masking bits and MACs, and the third round includes P_i performing offline evaluation of the circuit and generating output. Note that P_1 runs 2PC with all other parties at the same time in parallel. To implement secure broadcast, we use the simple echo broadcast [15, 23, 41]. Similar to previous work [40], we consider this broadcast to run in one round.

We now show how we divide our protocol into rounds, integrating 2PC and secure broadcasts. In the first two rounds of our protocol, we run the first two rounds of 2PC. As part of these two rounds, P_1 also broadcasts commitments to all their input bits and $\mathsf{Enc}(r_1)$ which is their share of r. In parallel, P_1 sends all $\mathsf{Enc}(s)$ from 2PC to FHE conversion, all Γ for ZK Protocol (3), and the $m = \mathsf{Enc}(\mu)$ of ZK Protocol (1) to each P_i, respectively, in parallel. Meanwhile, each P_i broadcasts their share $\mathsf{Enc}(r_i)$ and sends commitments $\mathsf{Com}(\sigma)$ for ZK Protocol (1) to P_1.

During our third round, P_i finishes the third round of 2PC. As soon as each P_i is done with 2PC evaluation, they open commitments to σs for P_1 and also send their Γs, Δs, Δ's, and ZK proofs.

In the fourth round, Party P_1 sends either μ or $\mu \oplus \iota$ of ZK Protocol (1) to P_i. Both parties broadcast a request to decrypt $\Delta'_{i,\pi(0)}$ and $\Delta'_{i,\pi(1)}$ such that P_1 learns the decrypted values. In parallel, P_i also broadcasts c'_i together with a ZK proof that this encrypts a bit.

In the fifth round, parties send their contributions to decrypt $\Delta'_{i,\pi(0)}$ and $\Delta'_{i,\pi(1)}$ to P_1 (together with a proof of correct decryption), and P_1 broadcasts c_i together with a proof that c_i encrypts a bit.

In the sixth and last round, all parties homomorphically compute c''_i, evaluate the arithmetic part of F and broadcast partial decryptions of their outputs together with a ZK proof of correct (partial) decryption.

Communication complexity: Wang et al.'s 2PC communication complexity is dominated by $O(1)$ hashes for each input and output wire. Thus, for evaluation of a single $f_{i,j,k}$ including 2PC to FHE conversion, we need to transmit $O(\ell \cdot (\log n + \lambda) + \lambda^2)$ hash values. For all comparisons, we therefore send $O(n \cdot (\ell \cdot (\log n + \lambda) + \lambda^2))$ hash values per party for the 2PC part.

For conversion, we first consider only communication between P_1 and P_i. More specifically, P_1 begins and sends 1 FHE encryption ($\mathsf{Enc}(s_1)$). For ZK Protocol (1), P_1 sends $O(\lambda\ell)$ ciphertexts $\mathsf{Enc}(m)$ to P_i and opens ciphertexts by sending as many random coins. Party P_i sends $O(\lambda\ell)$ hashes (commitments). For ZK Protocol (3), P_1 sends $O(\lambda)$ ciphertexts (Γs), P_2 also sends $O(\lambda)$ ciphertexts (their Γs) as well as Δ's and ZK proofs. Party P_i also sends 2 partial decryptions. For the last ZK Protocol (1), P_1 sends $O(\lambda^2)$ ciphertexts ($\mathsf{Enc}(m)$) and later opens with $O(\lambda^2)$ random coins, and P_i sends $O(\lambda^2)$ (hash) commitments.

In total, we send $O(n \cdot (\ell \cdot (\log n + \lambda) + \lambda^2)$ hashes per party. Assuming the size of a random coin and a ZK Proof to be like the size of a FHE ciphertext (up to constant factors), then P_1 sends $O(nd(\lambda\ell + \lambda^2))$ FHE ciphertexts. All other parties send significantly less ($O(n\lambda)$ ciphertexts).

Parties broadcast c_i and c'_i together with ZK proofs that these encrypt bits, leading to $O(dn)$ ciphertext broadcasts for P_1 and $O(n)$ for each P_i. So, we have a total number of of $O(dn)$ broadcasts and an (amortized) complexity of $O(n)$ broadcasts per party. In addition, each party broadcasts a single FHE ciphertext with $\mathsf{Enc}(r_i)$, and P_1 broadcasts ℓ commitments to their input, e.g., hash values. At the end of evaluating F, each party broadcasts a single partially decrypted FHE ciphertext and a ZK proof of correct decryption. With the size of this comparable to a FHE ciphertext up to constant factors, we have in total $O(n)$ FHE ciphertext broadcasts per party plus $O(\ell)$ hash values per party.