Mixed-Technique Multi-Party Computations Composed of Two-Party Computations

Erik-Oliver Blass[1] and Florian Kerschbaum[2]

[1] Airbus, Munich, Germany
erik-oliver.blass@airbus.com
[2] University of Waterloo, Waterloo, Canada
florian.kerschbaum@uwaterloo.ca

Abstract. Protocols for secure multi-party computation are commonly composed of different sub-protocols, combining techniques such as homomorphic encryption, secret or Boolean sharing, and garbled circuits. In this paper, we design a new class of multi-party computation protocols which themselves are composed out of two-party protocols. We integrate both types of compositions, compositions of fully homomorphic encryption and garbled circuits with compositions of multi-party protocols from two-party protocols. As a result, we can construct communication-efficient protocols for special problems. Furthermore, we show how to efficiently ensure the security of composed protocols against malicious adversaries by proving in zero-knowledge that conversions between individual techniques are correct. To demonstrate 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 computation 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 secure two-party computations are deployed in practice [68], 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 [33, 43, 44] based on Beaver et al.'s technique [5] 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) [11, 23] 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 high computation costs.

In this paper, we present a design alternative. We specifically consider multi-party computations that can at least partially be decomposed into a sequence of two-party computations (2PCs). We first evaluate 2PCs using garbled circuits and then combine the output and continue computation using FHE evaluation. 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 [13, 21, 29, 49]. For fully malicious security, we show how to convert between outputs of garbled circuits and FHE ciphertexts using efficient zero-knowledge proofs. Compared to conversions in the semi-honest model [39], this requires a different construction, which has, however, little additional overhead. Other related work [38] sketches malicious conversions, but only for two parties, whereas we consider the multi-party setting. The first phase of 2PC reduces multiplicative depth for the following FHE evaluation phase, but remains small enough to have low communication complexity. As we show by construction, such a combined protocol can keep a *constant number of rounds* and can still be 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 private set disjointness, i.e., a protocol that computes whether the intersection of sets is empty, but does not reveal anything else, including the intersection itself. This protocol follows a star topology of communication where each party P_i engages in 2PC with a central party P_1. Our composition of 2PC protocols into a multi-party protocol is particularly efficient if it follows a star topology. We stress that even in the star topology, we provide malicious security against an adversary controlling the central node (among others) which is the challenge of any such composition. Furthermore, besides the set disjointness protocol there are (infinitely) many other protocols that can be implemented in a star topology. The entire class of *multi-party private set analytics* protocols [4, 12, 20, 45, 51] is an example. However, our protocols are also not limited to a star topology, and we also mention other use cases, such as auctions [9], that do not follow a star topology.

Our example use case is 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. We have implemented a semi-honest version of this protocol to show that our design offers performance improvements over other multi-party computation protocols in the semi-honest model. Using our zero-knowledge proofs, our protocol can also be made secure in the malicious model.

In summary, the main contributions of this paper are:

1. A construction for **mixed-technique MPC** composed from 2PC which features a **constant number of rounds**, low communication complexity, and **malicious security**.
2. Efficient zero-knowledge proofs, included in this construction, converting between garbled circuit outputs and homomorphic encryption with malicious security.

3. A demonstration of our construction's usefulness by realizing a multi-party protocol for set disjointness.

We also present (Appendix D) a technique replacing standard verification of hash-based commitments during 2PC by a white-box use of garbled circuits. We use this technique to reduce communication overhead in our conversion, but the idea is general, applicable to other scenarios, and of independent interest.

2 Conversion between 2PC and Homomorphic Encryption

To simplify exposition, we start with a motivation and an overview of our conversion for the special case of $d = 2$ parties. For space reasons, we defer the extension to any $d \geq 2$ parties to Appendix B. Our goal is malicious security of the conversions which we describe in Section 2.1.

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 $O = (o_1, \ldots, o_N)$. 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. On the one hand, FHE evaluation of arithmetic circuits with large multiplicative depth is computationally expensive. On 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. Output of 2PC will serve as input to FHE and vice versa. 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}. 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.

2.1 Malicious Security

Achieving malicious security for conversion turns out to be a challenge. For example, let P_1 be the garbler and P_2 the evaluator during 2PC evaluation of a simple 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. Party P_2 should instead receive related information during 2PC which then allows P_2 to indirectly generate homomorphic encryptions $\mathsf{Enc}(x)$ and $\mathsf{Enc}(y)$. Alternatively, one might suggest implementing homomorphic encryption Enc inside a 2PC circuit, but this is too costly.

Similarly, we need to convert FHE ciphertexts output by 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, so they must prove to all parties that their encryptions are correct. Finally, the private key is shared among all d parties which impedes easy zero-knowledge (ZK) proofs.

Important Remarks This paper targets secure output conversion between 2PC and FHE. To actually evaluate Boolean sub-circuit 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 [41, 42, 53, 65] 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. To enforce this, our conversion requires the FHE scheme to also support distributed key generation and certain ZK proofs detailed below. There exist several efficient lattice-based FHE schemes with support for both [7, 8, 10, 17, 18, 50, 62], 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].

2.2 Solution Overview

Roadmap There are two different cases for conversion we will have to consider in a mixed-technique setting. First, parties 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 then evaluates the subsequent arithmetic circuit C_{i+1}^{Arith}.

Second, parties convert a sequence of ciphertexts $\mathsf{Enc}(b_i)$, homomorphic encryptions of bits b_i (or integers, see Appendix A) into input for a 2PC Boolean circuit evaluation. That is, both parties have evaluated arithmetic sub-circuit C_i^{Arith} and computed ciphertexts $\mathsf{Enc}(b_i)$, respectively. These ciphertexts will now be converted into input for 2PC evaluation of sub-circuit C_{i+1}^{Bool}.

Actual evaluation of circuits is then secure by definition, as we rely on standard maliciously-secure 2PC. For arithmetic sub-circuits, both parties evaluate FHE ciphertexts on their own. An honest party will automatically compute correct output ciphertexts as long as input ciphertexts are correct.

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. We only consider the first two cases of converting 2PC output to FHE input and FHE output to 2PC input.

Intuition Our conversions focus on Boolean sub-circuits C_i^{Bool}. We design mechanisms which either convert 2PC output of C_i^{Bool} to FHE ciphertexts serving as input to C_{i+1}^{Arith} or convert FHE ciphertexts coming from C_{i-1}^{Arith} into input to C_i^{Bool}. 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 for malicious security. 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 adapted, which is rather mechanical.

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, 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 $c_{1,i}$ 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_1 receives output. P_2 proves in ZK to P_1 that $c_{2,i}$ encrypts $\iota_{2,i}$, used during 2PC evaluation of γ. This is ZK Protocol (1) with roles of P_1 and P_2 reversed.
- 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.

Observe the different notation used in this paper for describing circuits. Boolean sub-circuits of function F are written as C_i^{Bool}, while Boolean circuits we use inside our ZK Protocol building blocks are written with the Greek letter γ.

Conversion The main idea behind the actual conversion is to modify a circuit C_i^{Bool} into γ which takes *shares* of C_i^{Bool}'s original input as its input and outputs shares of C_i^{Bool}'s original output. For example, to convert a 2PC output bit ω_1 of C_i^{Bool} to an FHE ciphertext $\mathsf{Enc}(\omega_1)$, we do not evaluate C_i^{Bool}, but γ which outputs share $\omega_1 \oplus s$ to P_2, and s to P_1. Both parties encrypt their shares, exchange resulting ciphertexts, and homomorphically compute an XOR to get $\mathsf{Enc}(\omega_1)$. During this conversion, ZK Protocols prove the correctness of operations.

So, 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.

Semi-Honest Security Our presentation concentrates on the case of fully malicious security. Nevertheless, even the semi-honest version of our conversion is of interest, as it enjoys the same properties as the fully-malicious version, e.g., $O(1)$ rounds, support for $d \geq 2$ parties, and moreover its performance is competitive when compared to related work, see Section 4.4. Essentially, the semi-honest version is just the fully-malicious one as described in the next section, but does not include the actual FHE ZK proofs inside ZK Protocols.

3 Technical Details

For simplicity, we keep describing details for $d = 2$ parties and extend to $d \geq 2$ parties in Appendix B.

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. 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. Also, this ensures that P_1 receives the correct output bit o'_j from P_2, i.e., ensure authenticity [6]. We silently rely on this trick for secure computation of all of P_1's plain output bits for the rest of the paper.

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. Later in Appendix D, we show how to efficiently realize commitments with a white-box use of wire labels in garbled circuits. 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, "$\cdot$" 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 $\pi : \{0,1\} \rightarrow \{0,1\}$. That is, $\Pi = \{\pi_0, \pi_1\}$ with $\pi_0(x) = x$ and $\pi_1(x) = 1 - x$.

3.1 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.

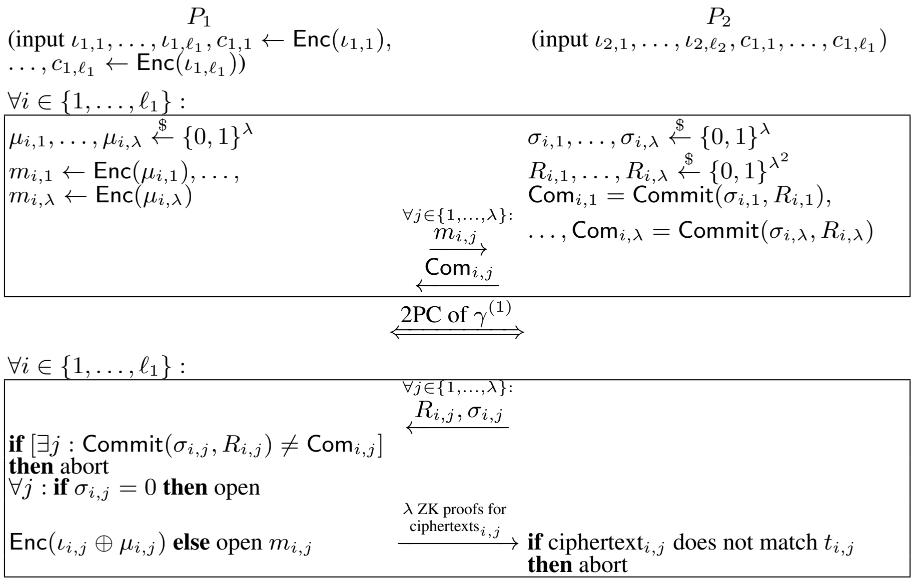

Fig. 1: ZK Protocol (1) for circuit γ

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 $\gamma^{(1)}$.

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, both parties use maliciously-secure 2PC and evaluate $\gamma^{(1)}$ which internally computes γ as a sub-routine. Party P_1 is the garbler and P_2 the evaluator. 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 $t_i = \mu_i$.

Output bit $\alpha = 0$ in $\gamma^{(1)}$ only serves to indicate protocol failure, i.e., non-matching commitments.

If $\sigma_{i,j} = 0$, then P_1 and P_2 homomorphically compute 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 ciphertext$_{i,j} = m_{i,j}$. Party P_1 then sends a ZK proof that ciphertext$_{i,j}$ 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, 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 $\gamma^{(1)}$

```
if ∀i, j, 1 ≤ i ≤ ℓ_1, 1 ≤ j ≤ λ : Com_{i,j} = Commit(σ_{i,j}, R_{i,j}) then
    α = 1;
    (ω_1, ..., ω_n) = γ((ι_{1,1}, ..., ι_{1,ℓ_1}), (ι_{2,1}, ..., ι_{2,ℓ_2}));
    for i = 1 to ℓ_1 and j = 1 to λ do
        if σ_{i,j} = 0 then t_{i,j} = ι_{1,i} ⊕ μ_{i,j} else t_{i,j} = μ_{i,j};
else α = ω_1 = ... = ω_n = t_{1,1} = ... = t_{ℓ_1,λ} = 0;
output α, ω_1, ..., ω_n, t_{1,1}, ..., t_{ℓ_1,λ};
```

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

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.

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. During 2PC, P_1 is the garbler and P_2 the evaluator. 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.

ZK Protocol (3) In this protocol, party P_2 proves to P_1 that encryptions $c_{\omega,i} \leftarrow \mathsf{Enc}(\omega_i)$ are encryptions of P_2's output bits ω_i. As ZK Protocol (3) is more involved, Figure 3 starts by presenting a slightly simpler version with a ZK proof which is only Honest-Verifier-Zero-Knowledge (HVZK), and details for fully-malicious security follow.

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.

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. 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.

The first bit of strings V_0, V_1 is output bit ω_i. That is, Γ_{ω_i} encrypts a bit string, where the first bit represents P_2's output bit ω_i. 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 proves 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 $\Gamma_2 = \mathsf{Enc}(V_{\omega_i})$, an encryption of the concatenation of P_2's

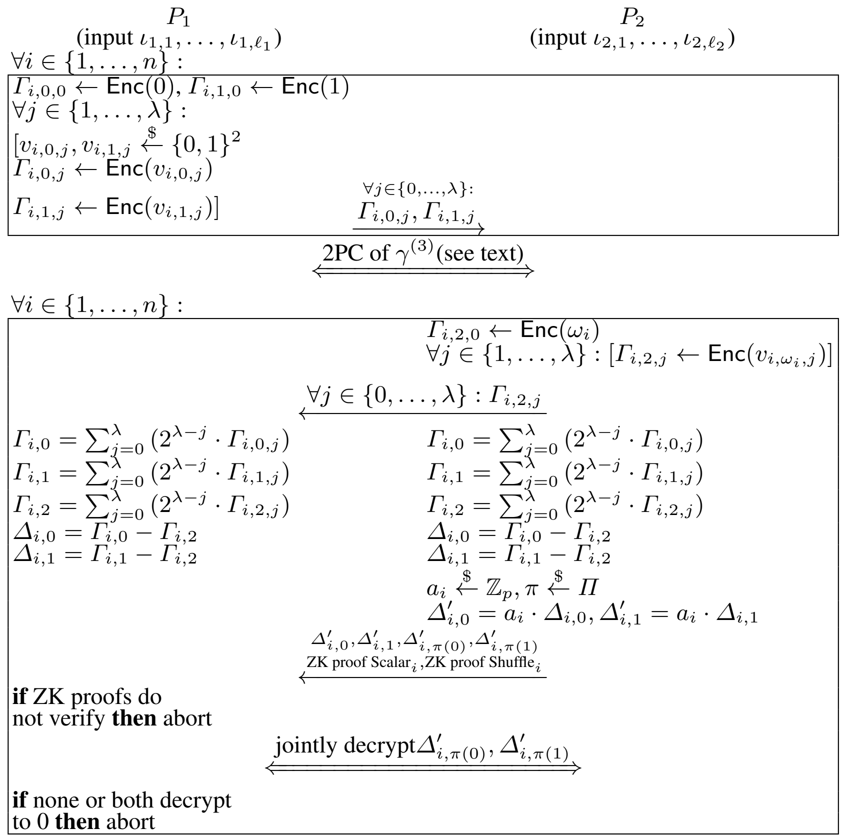

Fig. 3: ZK Protocol (3)

$\lambda + 1$ bits V_{ω_i}. As both parties know Γ_0 and Γ_1, they 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 (see below for details). 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, 7, 10].

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

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, P_2 blinds $\Delta_{i,0}$ and $\Delta_{i,1}$ by a random a_i resulting in $\Delta'_{i,0}$ and $\Delta'_{i,1}$. Then, P_2 prepares sub-ZK proof "Scalar$_i$" which proves that $\Delta'_{i,0}, \Delta'_{i,1}$ are the result of multiplying $\Delta_{i,0}, \Delta_{i,1}$ by the same secret scalar a_i. While such a proof is standard, e.g., P_2 could also simply publish the encryption of a_i, and P_1 computes $\Delta'_{i,0}, \Delta'_{i,1}$ themselves. Party 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

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

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

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 output** $\omega_i || v_{i,\omega_i,1}\cdots v_{i,\omega_i,\lambda}$;

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

of two-element shuffles are also 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 also implemented by, e.g., reverting to an efficient general proof [2] or by opening randomness of ciphertext $\beta - \beta^2$. 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.

HVZK to Fully-Malicious Security For fully-malicious security, 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 $\gamma^{(3)}$.

3.2 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 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. Thus, 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. All other necessary inputs $\mu_{i,j}$, $\sigma_{i,j}$, and $v_{\omega,j}$ are present for their respective input and outputs. This ensures 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. We can compose the conversion routines in a natural way. Figures 5 and 6 depict the details of the conversion from FHE to 2PC and reverse, respectively.

3.3 Security Analysis

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, maliciously secure protocol.

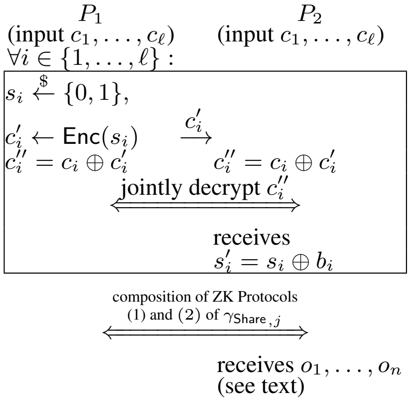

Fig. 5: FHE to 2PC conversion

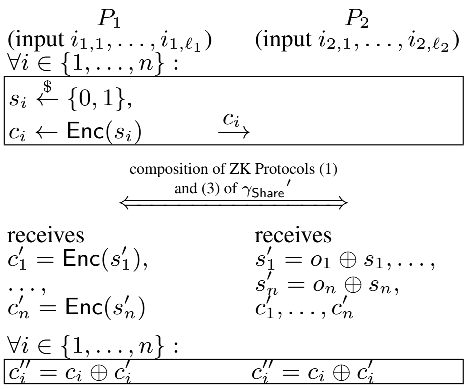

Fig. 6: 2PC to FHE conversion

Theorem 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., an 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.*

We prove Theorem 1 in Appendix C.

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 without revealing the intersection itself. While protocols computing PSD have been presented before [19, 22, 28, 35, 36, 46, 67], 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 [55, 57].

Each party P_i has an n element input set $S_i = \{e_{i,1}, \ldots, e_{i,n}\}$ with elements $e_{i,j} \in \{0, 1\}^\ell$. We present a protocol where parties securely compute whether the intersection of the S_i is empty, i.e., $|\bigcap_{i=1}^{d} S_i| \stackrel{?}{=} 0$. 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. So, 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 sub-circuit 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. [56] 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 [58, 60]. Each entry has space to store ℓ bits. Let $T_i[j,k]$ denote the k^{th} entry in the j^{th} 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 m^{th} 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:

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

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 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 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 $f_{i,j,k} = \left[\bigvee_{u=1}^{\beta}(T_1[j,k] \stackrel{?}{=} T_i[j,u])\right]$ with 2PC. For the k^{th} entry in their j^{th} 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 $r_i \stackrel{\$}{\leftarrow} M$ 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 $\mathsf{Enc}(F)$ which is then jointly decrypted. Without multiplying by r, parties would learn the size of the intersection.

4.2 Malicious Security for PSD

Although 2PC, 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. Consequently, we now show that adding our ZK protocols leads to a multi-party protocol secure in the malicious model, despite the fact that both parties of a two-party computation can be malicious (including the leader).

Recall that after 2PC to FHE conversion, 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 secure echo broadcast [25] 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.

An interesting situation 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 anyway 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 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 is then 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 $f_{i,j,k}$ anyway.

4.3 Complexity Analysis

Due to space constraints, we present and compare complexities of our mixed-techniques approach for evaluating F with related schemes in Appendix E.

4.4 Implementation

We have implemented our private set disjointness variant with 2PC to FHE conversion and performed micro-benchmarks. We will release our code into open source upon publication of the paper.

Our implementation of 2PC-part $f_{i,j,k}$ is done in the framework by Wang et al. [65] and maliciously secure. Yet, none of the common FHE libraries (HELib, PALISADE, SEAL, TFHE) provides both distributed key generation with threshold encryption and ZK proofs, which we need for maliciously-secure conversion. Moreover, an implementation of a FHE scheme with threshold decryption and ZK proofs, e.g., based on the

Table 1: Online time (s) to evaluate F, our scheme vs semi-honest and maliciously secure SPDZ vs 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 ("Semi-Malicious")				Semi-Honest		Malicious	
n	d	2PC	BC	FHE Comp	**Total**	**SPDZ^SH Total**	**FHE Total**	**SPDZ Total**	**BMR Total**
32	5	2.2	1.1	1.0	**4.3**	**10.1**	**141.7**	**16.4**	**8.5**
	10	3.9	1.8	1.8	**7.5**	**13.8**	**283.0**	**33.1**	**24.3**
	20	7.6	5.5	3.6	**16.6**	**48.8**	**565.5**	**50.3**	**Crash**
	40	14.8	17.6	7.1	**39.5**	**130.3**	**DNF**	**215.7**	**Crash**
64	5	4.7	1.4	2.3	**8.4**	**22.7**	**406.9**	**35.6**	**18.5**
	10	9.0	3.4	4.4	**16.8**	**32.6**	**813.1**	**72.4**	**66.6**
	20	18.0	10.7	8.6	**37.3**	**101.5**	**DNF**	**248.2**	**Crash**
	40	35.9	40.9	17.0	**93.8**	**265.8**	**DNF**	**784.3**	**Crash**
128	5	10.7	2.2	5.4	**18.3**	**52.3**	**DNF**	**117.5**	**43.0**
	10	20.8	6.6	10.3	**37.7**	**84.6**	**DNF**	**356.7**	**Crash**
	20	41.8	24.2	20.1	**86.1**	**358.1**	**DNF**	**675.8**	**Crash**
	40	83.3	95.3	39.7	**218.3**	**546.3**	**DNF**	**DNF**	**Crash**
1024	5	121.2	17.5	61.6	**200.4**	**727.3**	**DNF**	**DNF**	**DNF**
2048	5	265.0	37.5	135.5	**438.0**	**DNF**	**DNF**	**DNF**	**DNF**

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 (using TFHE [15, 16] for its simplicity), but not FHE ZK proofs, i.e., a semi-honest secure conversion. We dub the security setting of our implementation as "semi-malicious": 2PC is maliciously secure, but the conversion is only semi-honest secure. This setting is at least as strong as semi-honest security, but weaker than malicious security.

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).

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 $80,000$ circuits per second. However, network traffic, i.e., exchanging 177 KByte of data between P_1 and P_i during evaluation of that circuit, cannot be parallelized. Instead, we can only sequentially send 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 [64], we run benchmarks on one machine to better control network behavior and use `netem` [52] 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).

In Table 1, 2PC denotes the time to compute all $((\gamma_{\mathsf{Share}}'(1))(3))(1)$. BC denotes the time for all broadcasts of 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 (a partial decryption is one TFHE ciphertext). FHE Comp is the time, for each party, to compute the arithmetic part of F in TFHE.

For comparison, we have also implemented F in the popular MP-SPDZ framework [32] and benchmarked with both their semi-honest (SPDZ$^{\mathrm{SH}}$) and maliciously secure SPDZ variants as well as BMR [33]. SPDZ Total and BMR Total are their total (online) times to compute F. FHE Total is the total time of a semi-honest "pure-FHE" implementation of F with TFHE, including broadcasting each party's $m\beta\ell$ ciphertexts to all other parties. Note that BMR crashes even for a small number of parties, e.g., $n = 128, d = 10$, or quickly runs out of memory (> 32 GByte) for $d \geq 20$ parties.

Looking at Table 1, our implementation outperforms semi-honest and maliciously secure 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.

While timings for our "semi-malicious" implementation look promising regarding a potential maliciously secure implementation, we do not have such an implementation for the above stated reasons. However, observing that our techniques outperform even semi-honest SPDZ while offering stronger security guarantees leads to an interesting conclusion of our evaluation. Our mixed-techniques protocols might already serve as an alternative to standard semi-honest MPC in scenarios with a star topology, i.e., where a multi-party protocol can be decomposed into multiple 2PC protocols.

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 [37, 39]. Extending their conversion to also support fully-malicious adversaries is non-trivial: in Appendix D of [38], they present honest-verifier zero-knowledge proofs which render the protocol secure only if at most one party is malicious. However, HVZK is insufficient, if proofs are part of a 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 practical and efficient. Henecka et al. [27] design practical tools for conversion between garbled circuits and additively homomorphic encryption. Their conversion targets semi-honest adversaries and circuits for two parties. Demmler et al. [21] present a two party framework to convert between arithmetic sharing, Boolean sharing, and garbled circuits in the semi-honest model, and so do Riazi et al. [59]. Mohassel and Rindal [49] extend to three parties with malicious security. Again in the semi-honest model for two parties, Juvekar et al. [31] switch between garbled circuits and additively homomorphic encryption, and Büscher et al. [13] switch between arithmetic and Boolean sharing. Rotaru and Wood [61] and Aly et al. [1] convert between MPC based on arithmetic secret sharing and garbled circuits with malicious security.

For completeness sake, we mention that other powerful MPC frameworks besides SPDZ exist, e.g., the purely circuit-based EMP-Toolkit [66]. Also note that FHE is often combined with (arithmetic) MPC to prepare multiplication triplets during offline phases, as in, e.g., SPDZ and follow-up works [3, 34].

(Multi-Party) PSI and Disjointness While seminal works in PSI are based on dedicated protocols [47], recent papers use a circuit-based approach (see Pinkas et al. [54] for an overview), culminating in solutions with asymptotically optimal communication complexity and practical constants [57]. 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 [26] present a maliciously-secure multi-party PSI protocol based on oblivious polynomial evaluation (OPE). Similar to previous ideas [22], 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. [40] 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 (not disclosing the intersection) while maintaining efficiency [55, 57]. Other works have considered computing set disjointness, but these target semi-honest security and/or only two parties [19, 22, 28, 35, 36, 46, 67]

Comparing to related work, **our work** fills a gap with 1) a solution which converts between FHE and garbled circuits, 2) supports any number of parties, and 3) provides malicious security. We use this to present the first multi-party PSI analytics protocol whose communication complexity scales only quadratically in the number of participants d.

6 Conclusions

In this paper we have shown a new construction of secure multi-party computation techniques. We have shown i) how to combine them using multiple cryptographic techniques (garbled circuits and FHE), ii) how to combine them from two-party computations keeping communication cost low for important functions, such as private set analytics, and iii) how to make them secure against malicious adversaries. It is future work to implement and evaluate the runtime of our zero-knowledge protocols that make our construction maliciously secure. However, we analyzed its complexity and provided an implementation in the semi-honest model showing that our work outperforms existing approaches. The performance advantages stems from a reduction of the communication and round complexity which is critical for multi-party computations with many participants and their adoption in practice.

Bibliography

[1] A. Aly, E. Orsini, D. Rotaru, N.P. Smart, and T. Wood. Zaphod: Efficiently Combining LSSS and Garbled Circuits in SCALE. In *ACM WAHC*, 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 *EUROCRYPT*, 2012.
[3] C. Baum, D. Cozzo, and N.P. Smart. Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ. In *SAC*, 2019.
[4] A. Bay, Z. Erkin, M. Alishahi, and J. Vos. Multi-party private set intersection protocols for practical applications. In *SECRYPT*, 2021.
[5] D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols (Extended Abstract). In *ACM STOC*, 1990.
[6] M. Bellare, V.T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM CCS*, 2012.
[7] R. Bendlin and I. Damgård. Threshold Decryption and Zero-Knowledge Proofs for Lattice-Based Cryptosystems. In *TCC*, 2010.
[8] 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 *ASIACRYPT*, 2014.
[9] E.-O. Blass and F. Kerschbaum. Strain: A Secure Auction for Blockchains. In *ESORICS*, 2018.
[10] 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 *CRYPTO*, 2018.
[11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.
[12] P. Branco, N. Döttling, and S. Pu. Multiparty Cardinality Testing for Threshold Private Intersection. In *PKC*, 2021.
[13] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider. HyCC: Compilation of Hybrid Protocols for Practical Secure Computation. In *ACM CCS*, 2018.
[14] O. Catrina and S. de Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In *SCN*, 2010.
[15] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast Fully Homomorphic Encryption Library, 2016. https://tfhe.github.io/tfhe/.
[16] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptology*, 33(1), 2020.
[17] I. Damgård and A. López-Alt. Zero-Knowledge Proofs with Low Amortized Communication from Lattice Assumptions. In *SCN*, 2012.
[18] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO*, 2012.
[19] A. Davidson and C. Cid. An Efficient Toolkit for Computing Private Set Operations. In *ACISP*, 2017.
[20] S. Debnath, P. Stanica, N. Kundu, and T. Choudhury. Secure and efficient multiparty private set intersection cardinality. *Advances in Mathematics of Communications*, 15(2), 2021.
[21] D. Demmler, T. Schneider, and M. Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*, 2015.
[22] M.J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, 2004.
[23] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM STOC*, 2009.
[24] S. Goldfeder. A Boolean Circuit for SHA-256, 2019. `http://stevengoldfeder.com/projects/circuits/sha2circuit.html`.
[25] S. Goldwasser and Y. Lindell. Secure Multi-Party Computation without Agreement. *J. Cryptology*, 18(3), 2005.
[26] C. Hazay and M. Venkitasubramaniam. Scalable Multi-party Private Set-Intersection. In *PKC*, 2017.
[27] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM CCS*, 2010.
[28] S. Hohenberger and S.A. Weis. Honest-Verifier Private Disjointness Testing Without Random Oracles. In *PET*, 2006.
[29] M. Ishaq, A. Milanova, and V. Zikas. Efficient MPC via Program Analysis: A Framework for Efficient Optimal Mixing. In *ACM CCS*, 2019.
[30] M. Jawurek, F. Kerschbaum, and C. Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM CCS*, 2013.
[31] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security*, 2018.

[32] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. IACR ePrint 2020/521, 2020.
[33] M. Keller and A. Yanai. Efficient Maliciously Secure Multiparty Computation for RAM. In *EUROCRYPT*, 2018.
[34] M. Keller, V. Pastro, and D. Rotaru. Overdrive: Making SPDZ Great Again. In *EUROCRYPT*, 2018.
[35] A. Kiayias and A. Mitrofanova. Testing Disjointness of Private Datasets. In *FC*, 2005.
[36] L. Kissner and D. Xiaodong Song. Privacy-Preserving Set Operations. In *CRYPTO*, 2005.
[37] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *CANS*, 2009.
[38] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. From dust to dawn: Practically efficient two-party secure function evaluation protocols and their modular design. IACR ePrint 2010/079, 2010.
[39] 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. *J. Computer Security*, 21(2), 2013.
[40] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. Practical Multi-party Private Set Intersection from Symmetric-Key Techniques. In *ACM CCS*, 2017.
[41] V. Kolesnikov, J.B. Nielsen, M. Rosulek, N. Trieu, and R. Trifiletti. DUPLO: Unifying Cut-and-Choose for Garbled Circuits. In *ACM CCS*, 2017.
[42] Y. Lindell. Fast Cut-and-Choose-Based Protocols for Malicious and Covert Adversaries. *J. Cryptology*, 29(2), 2016.
[43] Y. Lindell, N.P. Smart, and E. Soria-Vazquez. More Efficient Constant-Round Multi-party Computation from BMR and SHE. In *TCC*, 2016.
[44] Y. Lindell, B. Pinkas, N.P. Smart, and A. Yanai. Efficient Constant-Round Multi-party Computation Combining BMR and SPDZ. *J. Cryptology*, 32(3), 2019.
[45] R. Akhavan Mahdavi, T. Humphries, B. Kacsmar, S. Krastnikov, N. Lukas, J. Premkumar, M. Shafieinejad, S. Oyan, F. Kerschbaum, and E.-O. Blass. Practical Over-Threshold Multi-Party Private Set Intersection. In *ACSAC*, 2020.
[46] L. Marconi, M. Conti, and R. Di Pietro. CED^2: Communication Efficient Disjointness Decision. In *SECURECOMM*, 2010.
[47] C.A. Meadows. A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party. In *IEEE S&P*, 1986.
[48] C. Aguilar Melchor, M.-O. Killijian, C. Lefebvre, and T. Ricosset. A Comparison of the Homomorphic Encryption Libraries HElib, SEAL and FV-NFLlib. In *SecITC 2018*, 2018.
[49] P. Mohassel and P. Rindal. ABY^3: A Mixed Protocol Framework for Machine Learning. In *ACM CCS*, 2018.
[50] S. Myers, M. Sergi, and A. Shelat. Threshold Fully Homomorphic Encryption and Secure Computation. IACR ePrint 2011/454, 2011.
[51] S. Narayanan, T. Aishwarya, A. Agrawal, A. Patra, A. Choudhary, and P. Rangan. Multi party distributed private matching, set disjointness and cardinality of set intersection with information theoretic security. In *CANS*, 2009.
[52] Netem. netem, 2019. `https://wiki.linuxfoundation.org/networking/netem`.
[53] J.B. Nielsen and C. Orlandi. LEGO for Two-Party Secure Computation. In *TCC*, 2009.
[54] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security*, 2015.
[55] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT*, 2018.
[56] B. Pinkas, T. Schneider, and M. Zohner. Scalable Private Set Intersection Based on OT Extension. *ACM Trans. Priv. Secur.*, 21(2), 2018.
[57] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient Circuit-Based PSI with Linear Communication. In *EUROCRYPT*, 2019.
[58] M. Raab and A. Steger. "Balls into Bins" - A Simple and Tight Analysis. In *RANDOM*, 1998.
[59] 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 *ACM AsiaCCS*, 2018.
[60] P. Rindal and M. Rosulek. Malicious-Secure Private Set Intersection via Dual Execution. In *ACM CCS*, 2017.
[61] D. Rotaru and T. Wood. MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security. In *INDOCRYPT*, 2019.

[62] M. Strand. A Verifiable Shuffle for the GSW Cryptosystem. In *VOTING*, 2018.
[63] M. Varia, S. Yakoubov, and Y. Yang. HEtest: A Homomorphic Encryption Testing Framework. In *WAHC*, 2015.
[64] Verizon. IP Latency Statistics, 2020. `https://enterprise.verizon.com/terms/latency/`.
[65] X. Wang, S. Ranellucci, and J. Katz. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *ACM CCS*, 2017.
[66] X. Wang, S. Ranellucci, and J. Katz. Global-Scale Secure Multiparty Computation. In *ACM CCS*, 2017.
[67] Q. Ye, H. Wang, J. Pieprzyk, and X.-M. Zhang. Efficient Disjointness Tests for Private Datasets. In *ACISP*, 2008.
[68] M. Yung. From Mental Poker to Core Business: Why and How to Deploy Secure Computation Protocols? In *ACM CCS*, 2015.

A 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 anyway 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 $\sum_{i=0}^{n-1} 2^i \cdot c_{i+1}$.

B $d \geq 2$ Parties

Secure multi-party computation can be constructed from secure two-party computations in various ways. One standard way is a star topology as we will present in our example in Section 4. We emphasize, however, that our conversions are not limited to star topologies.

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.

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 [9]. 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 would 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. We use secure two-party protocols during the first phase to achieve efficient implementations in a constant number of (communication) rounds. Similarly, to evaluate low multiplicative depth sub-circuits, we use homomorphic encryption efficiently. Our ZK protocols ensure that the conversion is secure against malicious adversaries.

C Proof of Theorem 1

We emphasize that we only provide a proof-sketch that, however, should convince an expert reader about the correctness of our theorems and the security of our protocols. Before presenting this proof sketch of our main Theorem 1, we briefly recall completeness, zero-knowledge, and soundness definitions.

Let $P \in \{P_1, P_2\}$ be the prover and $V \in \{P_1, P_2\}$ be the verifier in a ZKP. Let $w \in R_C$ be a witness for the correct execution of a conversion which we denote as relation R_C. Let $\langle P(w), V\rangle$ be the execution of a ZKP protocol.

Completeness: An honest verifier accepts the proof, if the prover provides consistent input, that is:

$$w \in R_C \Longrightarrow \langle P(w), V\rangle \wedge Pr[V = \mathsf{accept}] = 1$$

Zero-Knowledge: The verifier learns nothing about the prover's witness except that it satisfies the proof, i.e., there exists a simulator Sim_P such that:

$$\langle P(w), V\rangle \stackrel{c}{=} \langle \mathsf{Sim}_P, V\rangle$$

Soundness: An 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, i.e., there exists an extractor Ext_V such that:

$$V = \mathsf{accept} \Longrightarrow \langle P(w), \mathsf{Ext}_V\rangle \wedge Pr[\mathsf{Ext}_V = w] = 1 - \mathsf{negl}(\lambda)$$

Proof (Theorem 1).

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 Sim_{P_1} or Sim_{P_2} in the hybrid model which 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 Sim_{P_1}, or Sim_{P_2} (in ZKP (2)), 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 Sim_{P_1} 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 Ext_{P_1} or Ext_{P_2}. We construct an extractor Ext_{P_2} only for ZKP (1), but stress that the extractor Ext_{P_1} 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 [6], and we do not need an extractor. 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). □

D Replacing Hash-based Commitments

In compositions of multi-party protocols from two-party protocols, an important application of our conversions can be used which is of independent interest. In general, 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 hashes during 2PC. This requires about 22000 AND gates for each 256 input bits using, for example, SHA2 [24]. Our construction below omits hash verification inside the circuit and can be used as an alternative.

Details We describe how this technique is applied to ZK Protocols (1) and (2), but stress that it is general and can be applied in other scenarios, too. More specifically, the costliest operation during garbled circuit 2PC evaluation in ZK Protocols (1) and (2) is

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, $\mathcal{C}_{GF(2^{\ell+\lambda})*}$: 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: $O(n\ell) \cdot BC_\kappa \subseteq O(\mathcal{I}) \cdot BC_{|\mathsf{FHE}|}$, $\ell d^3 \in O(d\mathcal{I})$, $\frac{\lambda \mathcal{I}}{\log n} + dn\lambda^2 \in O(\mathcal{I})$, $O(I + nd\lambda \cdot (\ell + \lambda)) \cdot H \subseteq O(\mathcal{I}) \cdot \mathcal{C}_{\mathsf{FHE}*}$, $O(n \cdot (\ell \cdot (\log n + \lambda) + \lambda^2)) \cdot |H| \subseteq O(\mathcal{I}) \cdot |\mathsf{FHE}|$, $O(nd(\lambda\ell + \lambda^2)) \cdot |\mathsf{FHE}| \subseteq O(\mathcal{I}) \cdot |\mathsf{FHE}|$, $O(\ell) \cdot BC_{|H|} \subseteq O(nd) \cdot BC_{|\mathsf{FHE}|}$.

	Comp. / party	Comm. / party	Rounds
FHE	$O(\mathcal{I}) \cdot \mathcal{C}_{\mathsf{FHE}*}$	$O(\ell n) \cdot BC_{|\mathsf{FHE}|}$	$O(1)$
Constant Round MPC [43]	$O(\mathcal{I}) \cdot \mathcal{C}_{\mathsf{FHE}*}$	$O(\mathcal{I}) \cdot BC_{|\mathsf{FHE}|}$	$O(1)$
SPDZ [18]	$O(d\mathcal{I}) \cdot \mathcal{C}_{GF(2^{\ell+\lambda})*}$	$O(d\mathcal{I}) \cdot |\mathsf{SYM}| + O(n) \cdot BC_{|GF(2^{\ell+\lambda})|}$	$O(\log d + \log\log n)$
This paper	$O(\mathcal{I}) \cdot \mathcal{C}_{\mathsf{FHE}*}$	$O(\mathcal{I}) \cdot |\mathsf{FHE}| + O(n) \cdot BC_{|\mathsf{FHE}|}$	$O(1)$

verification of commitments $\mathsf{Com}_{i,j}$. For hash-based commitments, $\gamma^{(1)}$ and $\gamma^{(2)}$ would need to comprise sub-circuits recomputing expensive 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 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. [30], 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.

E Complexity Analysis

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 [18], constant-round MPC [43], 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 [18, 25, 43, 44] which has message complexity $O(d^2)$.

(Semi-Honest) FHE Let $\mathcal{C}_{\mathsf{FHE}*}$ be the computational complexity for a FHE multiplication and $\mathcal{C}_{\mathsf{FHE}+}$ the computational complexity 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 $O(d\ell n \log n) \cdot \mathcal{C}_{\mathsf{FHE}*}$ homomorphic multiplications with a multiplicative circuit depth of $\log \ell + \log\log n + \log d + 1$. Even for reasonable values $\ell = 32$, $d = 20, n = 128$, the multiplicative depth is already 14 which leads to huge runtimes in practice [48]. Note that homomorphic additions also increase ciphertext noise. While noise increased by additions is roughly one order of magnitude less than with multiplications [63], 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 [33, 43, 44] 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. [43] require 9 rounds and a FHE multiplicative depth of 3.

SPDZ Comparing two ℓ Bit integers is implemented in SPDZ [18] by Catrina and de Hoogh [14]'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

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	$\ell \log n$
$\gamma_{\mathsf{Share}}{}'$	$+1, +0$	$+0$	$+1$
$\gamma_{\mathsf{Share}}{}'(1)$	$+\lambda\ell, +\lambda\ell$	$+\lambda\ell$ (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	$\ell \cdot (\log n + \lambda) + \lambda^2$	$\lambda\ell + \lambda^2$	$\ell \cdot (\log n + \lambda) + \lambda^2$

exchanged between two parties. Initial sharing of $O(n)$ elements of each party requires $O(n)$ secure broadcasts.

Our Mixed-Technique Let $\mathcal{C}_{\mathsf{2PC}}$ 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 $O(nd) \cdot \mathcal{C}_{\mathsf{2PC}}$ plus $O(nd) \cdot \mathcal{C}_{\mathsf{FHE}*}$ plus $O(n) \cdot \mathcal{C}_{\mathsf{FHE}+}$. $m \in O(\frac{n}{\log n})$ and $\beta \in O(\log n)$, So, the computational complexity is in $O(n \cdot (\mathcal{C}_{\mathsf{FHE}+} + d \cdot (\mathcal{C}_{\mathsf{2PC}} + \mathcal{C}_{\mathsf{FHE}*})))$.

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 $\gamma_{\mathsf{Share}}{}'$ of $f_{i,j,k}$, which adds additional complexity, see also Table 3. Specifically, running $\gamma_{\mathsf{Share}}{}'$ 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 $\gamma_{\mathsf{Share}}{}'$ 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.

We use the scheme by Wang et al. [65] as 2PC building block which implements evaluation of each circuit with $O(\ell \cdot (\log n + \lambda) + \lambda^2)$ calls to a cryptographic hash function. There are d parties and $O(\frac{n}{\log n})$ hash table buckets of $O(\log n)$ entries, leading to a total of $O(n \cdot (\ell \cdot (\log n + \lambda) + \lambda^2))$ hashes per party.

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 [65]) 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 [18, 25, 44]. Similar to previous work [43], 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).