# Assisted Multi-Party Computation

Philipp Muth[1] and Stefan Katzenbeisser[2]

[1] TU Darmstadt, Department of Computer Science
philipp.muth@tu-darmstadt.de
[2] Universität Passau, Faculty for Computer Science and Mathematics

**Abstract.** Since their introduction in the 1970s, multi-party computation protocols have become the prevalent method for two or more parties to jointly compute an agreed upon function on private inputs without revealing them to other parties. While some efficiency gains in the offline phase of MPC protocols have been achieved, most works in the past have focused on optimising the online phase. Improvements to the online phase typically shifted significant workload to the offline phase. In this work we explore a novel approach to streamline the offline phase of secret sharing based MPC protocols by introducing a helper party that executes the preprocessing for the parties engaged in the online phase. We prove, that the security guarantees provided by the MPC protocols stay unchanged and demonstrate the efficiency of our approach in two sets of benchmarks. We furthermore give three examples of real world instantiations of the helper party to demonstrate that our approach is not only of a theoretical nature.

**Keywords:** Multi-party computation · Secret sharing · Preprocessing speed up

## 1 Introduction

### 1.1 Motivation

The privacy of data has been a concern to the earliest societies that communicated via literary language. From military communication to trade secrets or medical information confidentiality has been of utmost importance. This need has seamlessly carried over to the digital age with the amount of data being generated, transmitted and stored steadily increasing and containing more and more personal details. In fact, with eavesdroppers' – be that statefunded or private – computational power and cryptoanalytical capabilities improving perpetually, the need for privacy has rarely been more dire.

The introduction of computers has not only improved attackers' capabilities but also has grown the field of applications with a need for privacy. It is no longer restricted to protecting data in communication and storage, it also includes computation on private data. Especially when computing on data from multiple sources, maintaining confidentiality yields a pressing challenge.

For this scenario, the first so-called secure multi-party computation (MPC) protocols were introduced in the 1970s. An MPC protocol enables a set of parties to jointly evaluate an agreed upon function represented as a circuit, so that each party provides a private input, the parties jointly execute the computation and a result is produced while each party learns only the result and what can be derived from the output with respect to the other parties' inputs.

Garbled circuits and secret sharing based protocols are the two most important categories of MPC protocols. The most prominent example of the former category is Yao's garbled circuit [26]. The latter contains protocols, in which inputs to a circuit are distributed among the parties via a secret sharing scheme. The circuit is evaluated layer by layer in communication rounds and the result is jointly opened by publishing the shares of the result. In this work we focus on secret sharing based MPC protocols.

Secret sharing based MPC protocols comprise of an offline or preprocessing phase and an online phase. In the offline phase, the engaged parties jointly generate and share the auxiliary data necessary for the evaluation of the circuit. For example, many secret sharing based MPC protocols use random "Beaver triples" for performing multiplications, that are computed in the offline phase. The offline phase is executed independently of the parties' inputs, it solely depends on the structure of the circuit. In the online phase the actual evaluation of the circuit takes place. The parties provide their respective private inputs and jointly compute the output of each gate to arrive at the final result.

In recent years, the focus of most research aiming to improve the efficiency of MPC protocols has been on optimising the online phase, and remarkable achievements have been made. Yet in most cases, these improvements to the online phase entail performance penalties in the offline phase. While preprocessing can be executed far in advance to the online phase and is independent of the parties' inputs, it can however not be skipped. An example, that illustrates the performance disparity between offline and online phase, is the minimum euclidean distance computation for 1000 values of 128 bits in the framework ABY [10] between two parties with a network delay of 50ms. The time for the online phase amounts to 4442ms, whereas the offline phase takes 16506ms, i.e., close to 80 percent of the protocol's duration was spent on preprocessing.

### 1.2 Our contribution

In this work we propose an approach to significantly improve the performance of the offline phase for secret sharing based MPC protocols. We achieve this by introducing an independent helper party that assists in executing the offline phase and sharing the resulting data among the original parties. The helper party will not take part in the online phase, it can hence not obtain any knowledge with respect to the parties' inputs or the execution of the online phase. Indeed, we show that the introduction of the helper party does not affect the security guarantees provided by the original MPC protocol. We furthermore demonstrate the practical applicability of our approach by giving an implementation of the helper party for the SPDZ offline phase. We empirically test the efficiency of our

implementation in two scenarios. The first is the rate of Beaver triples generated and shared per second among two to five parties. The second is the preprocessing for a Vickrey auction with one hundred bids. Our tests show a performance improvement of up to a factor of 69 in comparison to previous works like SPDZ [8] and evolutions thereof. We discuss three real world instantiations of our approach. We elaborate on their respective limitations and advantages regarding performance. We thereby demonstrate that our approach is not of a purely theoretical nature, but can in fact be deployed in real executions of secret sharing based MPC protocols. Our approach is applicable to any MPC protocol based on secret sharing.

### 1.3 Structure

First, we present our approach to improve the efficiency of the offline phase in Section 3, that is an additional helper party $P_h$, that takes on executing the offline phase and distributing the resulting shares to the original parties of the MPC protocol. Second, we prove that if the original protocol was simulatable, the resulting protocol with $P_h$ executing the offline phase is simulatable as well. Third, we discuss how our approach can be applied to SPDZ [8] in Section 4.1. Fourth, we give benchmark results for our implementation of the helper party in the context of SPDZ in Section 4.2, that show a significant performance benefit from our approach in comparison to the unmodified protocols. Finally we discuss feasible real world implementations of our model in Section 5.

### 1.4 Related work

Secret sharing schemes were first introduced independently in the 1970s by Shamir [23] and Blakley [3]. Both proposed threshold secret sharing schemes with perfect secrecy over the ring $\mathbb{Z}_p$ for a prime $p$ larger than the number of shareholders. Verifiable secret sharing was developed as an additional feature in works like [20] and [14]. Fitzi et al. [13] presented a different approach to verifiable secret sharing, that provided a storage efficient scheme in three rounds to share a secret. Tassa [24] introduced an extension of Shamir's scheme, that enables a multi-level threshold access structures, so that for reconstruction the threshold of each hierarchy level has to be fulfilled. Traverso et al. [25] further improved the capabilities of hierarchical secret sharing by proposing a scheme that was also dynamic and verifiable. Our approach is applicable to any secret sharing based MPC protocol, independent of the underlying secret sharing scheme.

A scheme enabling integer secrets outside $\mathbb{Z}_p$ was later presented by Damgard and Thorbek [9] along with a protocol for distributed exponentiation on the shared secret. Its caveat is that the security is only computational. Rabin and Ben-Or [21] combined verifiable secret sharing with general computation on shared secrets, thus a more general approach, yet less performant.

The field of secure multi-party computation holds a wide choice of schemes that have been established over the years, most prominent of which are Goldreich et al.'s GMW [15] and Damgard et al.'s SPDZ [8]. Both schemes rely on computational assumptions with respect to security, whereas Bogdanov et al. [4] provide an entirely information theoretically secure approach with Sharemind at the cost of performance. Cramer et al. [6] transposed SPDZ to a setting where computation is not over finite fields $\mathbb{F}_{p^k}$ but over the more practical binary field $\mathbb{F}_{2^k}$, providing the same security guarantees. We will show, that our proposition maintains the security guarantees of the MPC protocol, that it is being applied to, while yielding a significant performance boost to the offline phase.

A secure two-party computation protocol employing a third party for the circuit evaluation was discussed by Feige et al. [11]. We continue this direction of research by employing the additional party exclusively for preprocessing, thereby strengthening the security properties of their approach, since we neither introduce nor rely on further computational hardness assumptions.

The approach of outsourcing the offline phase to a smaller set of parties – either disjoint or a subset of the original set of parties – was discussed previously by Scholl et al. [22]. A different direction was taken in Keller et al.'s MASCOT [16], where the offline phase was improved by employing oblivious transfer protocols (OT). In Overdrive, Keller et al. [17] proposed that SPDZ with improvements on its original design provides an offline phase that is similar to the improvements achieved by MASCOT. With Overdrive2k, Orsini et al. [19] transferred the improvements of Overdrive to the setting of computation over $F_{2^k}$. Their works engage all parties of the online phase in the offline phase, whereas we have the helper party $P_h$ execute the offline phase. We test the performance of our approach on the offline phase of SPDZ.

## 2 Preliminaries

### 2.1 Notation

Throughout this work we will use the following notational conventions. For an algorithm $\mathcal{A}$, we denote its output $y$ upon receiving input $x$ by $y \leftarrow \mathcal{A}(x)$. If $\mathcal{A}$ is a probabilistic algorithm, we write $y \leftarrow\!\!\$\ \mathcal{A}(x)$.

For a set $X$, we denote the order, that is the number of elements contained in it, by $\#X$.

Let $D_0$ and $D_1$ be two random variables sampling from the same set $X$. For an algorithm $\mathcal{A}$, we denote the advantage in distinguishing $D_0$ and $D_1$ by

$$\mathsf{Adv}^{\mathrm{dist}}_{D_0,D_1}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{\mathrm{dist}}_{D_0,D_1}(\mathcal{A}) = 1\right] - \frac{1}{2}\right|.$$

If, for any algorithm $\mathcal{A}$, we have $\mathsf{Adv}^{\mathrm{dist}}_{D_0,D_1}(\mathcal{A}) = 0$, then we call $D_0$ and $D_1$ *perfectly indistinguishable* and write

$$D_0 \overset{\mathrm{perf}}{=} D_1.$$

$\underline{\mathsf{Exp}^{\mathrm{dist}}_{D_0,D_1}(\mathcal{A})}$

$b \leftarrow\!\!\$\ \{0,1\}$

$b' \leftarrow \mathcal{A}^{\mathcal{O}_{D_b}}$

**return** $b == b'$

$\underline{\mathcal{O}_{D_b}()}$

$x \leftarrow\!\!\$\ D_b$

**return** $x$

**Fig. 1.** Experiment $\mathsf{Exp}^{\mathrm{dist}}_{D_0,D_1}(\mathcal{A})$

### 2.2 Secret Sharing Schemes

A secret sharing scheme is a cryptographic primitive that enables a dealer $\mathcal{D}$ to distribute a secret from a given secret space among a set of shareholders $\{S_1, \ldots, S_n\}$. A subset of parties that can reconstruct a shared secret from their respective shares is called authorised. The set of all authorised sets of parties is called an access structure, which we denote by $\Gamma \subset 2^{\{S_1,\ldots,S_n\}}$. An instance $\mathcal{S}$ of a secret sharing scheme is thus defined by the set of shareholders $\{S_1, \ldots, S_n\}$, the secret space and the access structure $\Gamma$.

A secret sharing scheme provides two protocols `share` and `recon`. In an instance $\mathcal{S}$, the dealer $\mathcal{D}$ executes $\mathcal{S}.\mathtt{share}(s)$ on a secret $s$ to obtain shares $\{s_1, \ldots, s_k\}$. These shares are then assigned and distributed to the shareholders according to a surjective map $\psi : \{1, \ldots, k\} \to \{1, \ldots, n\}$, i.e., shareholder $S_i$ receives all shares $s_j \in \{s_1, \ldots, s_k\}$ with $\psi(j) = i$. An authorised set $S \in \Gamma$ can then execute $\mathtt{recon}\Big(\{s_j\}_{S_{\psi(j)} \in S}\Big)$ on their respective shares to reconstruct a previously shared secret.

In this work we consider information theoretic secret sharing schemes, that is to any unauthorised set of shareholders $S' \notin \Gamma$, and any distinct secrets $s \neq s'$,

$$\Pr\Big[\{s_i\}_{S_{\psi(i)} \in S'} \subset \mathcal{S}.\mathtt{share}(s)\Big] = \Pr\Big[\{s_i\}_{S_{\psi(i)} \in S'} \subset \mathcal{S}.\mathtt{share}(s')\Big]$$

holds.

### 2.3 MPC Protocols

A multi-party computation (MPC) protocol is an algorithm, that – when executed correctly – enables two or more parties to jointly evaluate a prescribed function on their respective inputs. In this work, we focus on secret sharing based MPC protocols. That is, we consider protocols that evaluate arithmetic circuits $C_f$ representing a function

$$f : \mathbb{F}^n \to \mathbb{F}^n; (x_1, \ldots, x_n) \mapsto (y_1, \ldots, y_n)$$

for a field $\mathbb{F}$, where $x_i$ is a party $P_i$'s input and $y_i$ its output for $i = 1, \ldots, n$.

The execution of a secret sharing based MPC protocol is handled in two phases: the offline or preprocessing phase and the online phase. In the offline

phase the parties generate and share the auxiliary data necessary for the evaluation of the circuit $C_f$. This includes but is not limited to the randomness used throughout the online phase and additional shared data for the evaluation of individual gates such as Beaver triples. The data generated in the offline phase is independent of the inputs, that the parties provide in the online phase. Therefore significant time can elapse between the execution of the offline phase and the online phase. In the online phase the circuit $C_f$ is evaluated with the parties' inputs utilising the data generated during the offline phase. Each player receives his output according to the prespecified output gates.

A multi-party computation protocol has two main aims: correctness and privacy. Correctness implies, that for any input $(x_1, \ldots, x_n)$ the MPC protocol outputs $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ to the parties engaged in the online phase. Privacy means, that the protocol must avoid divulging more information to a party $P_i$ than it could otherwise derive from its input $x_i$ and the output $y_i$. For that, we define the view of a party $P_i$ as its input, the randomness used in the online phase and the messages it received, i.e.,

$$\mathsf{view}_i := \left\{ x_i, (r_i)_j, (m_i)_j \right\}.$$

Since the view of a party $P_i$ contains $(r_i)_j$ the randomness, that $P_i$ uses in the online phase, and the messages that it receives from the other parties, $\mathsf{view}_i$ is a random variable.

If a view of an unauthorised set of parties, that is indistinguishable from a real view, can be produced from those parties' inputs and outputs, we have perfect privacy, since no information can be derived from the knowledge gained during the online phase.

**Definition 1 (Simulatability).** *We call an MPC protocol* simulatable*, if there exists an efficient algorithm* $\mathsf{Sim}$*, that for any unauthorised set* $C \subset \{P_1, \ldots, P_n\}$ *upon input* $\{x_i, y_i\}_{P_i \in C}$ *produces an output, that is perfectly indistinguishable from the real view of* $C$*, i.e. for any adversary* $\mathcal{A}$*, we have*

$$\mathsf{Adv}^{dist}_{\mathsf{Sim}\left(\{x_i, y_i\}_{P_i \in C}\right), \{\mathsf{view}_i\}_{P_i \in C}}(\mathcal{A}) = 0.$$

### 2.4 Adversary

We consider a static and active attacker. That is an attacker, that upon the initialisation of the protocol corrupts an unauthorised subset $C \subset \{P_1, \ldots, P_n\}$ of the shareholders. Throughout the execution of the MPC protocol, this set cannot be changed. The adversary obtains all knowledge, that the corrupted shareholders have. This includes their inputs $\{x_i\}_{P_i \in C}$, the randomness used by the shareholders $\{(r_i)_j\}_{P_i \in C}$ and the messages they receive from the other parties in the protocol $\{(m_i)_j\}_{P_i \in C}$. The adversary controls all outputs of the corrupted parties and the messages, that they send to other parties.

## 3 Model

In the offline phase of an MPC protocol, the parties $P_1, \ldots, P_n$ jointly generate data, that is used in the online phase to evaluate $C_f$. This data is independent of the inputs, that $P_1, \ldots, P_n$ provide in the online phase. The structure of the data is determined by the gates contained in $C_f$. For an arbitrary but fixed circuit $C_f$, let $\{t_1, \ldots, t_m\}$ therefore be the set of all types of gate contained in $C_f$. In an arithmetic circuit, $t_1$ may for example denote input gates, $t_2$ multiplication gates, $t_3$ addition gates and $t_4$ output gates. We denote the set of all gates in $C_f$ by $\mathcal{G} = \{g_1, \ldots, g_{|C_f|}\}$. Furthermore, we define a function $\phi : \mathcal{G} \to \{t_1, \ldots, t_m\}$, that maps a gate to its type. In the offline phase, the parties $P_1, \ldots, P_n$ jointly sample a data set $D = \{d_1, \ldots, d_{|C_f|}\}$, where $d_i$ contains the data necessary for the evaluation of the gate $g_i$, $i = 1, \ldots, |C_f|$. Let $D_1, \ldots, D_m$ be a partition of $D$ faithful to the gates' type, that is

$$\bigcup_{i=1}^{m} D_i = D \tag{3.1}$$

holds, while $D_i \cap D_j = \emptyset$ for all $1 \leq i < j \leq m$, as well as

$$\forall i = 1, \ldots, m : \forall d, d' \in D_i : \phi(d) = \phi(d').$$

For each $D_i$, $1 \leq i \leq m$, there exists an underlying distribution $X_i$ from which $D_i$ is sampled.

We introduce a new party to the MPC protocol. This party we call the "helper party" $P_h$. Its purpose is to execute the offline phase in place of the parties $P_1, \ldots, P_n$ in order to give a significant speed up compared to the traditional execution. The parties $P_1, \ldots, P_n$ give a description of the circuit $C_f$ and a probability $p_f > 0$, that indicates the trust they have in $P_h$, as input to $P_h$. We further elaborate on $p_f$ later on. From the descrciption of $C_f$, $P_h$ derives the set of data $D$ according to (3.1), that is to be produced in the offline phase. $P_h$ generates and shares a data set corresponding to the distribution of $D$, thereby executing the offline phase in place of $P_1, \ldots, P_n$. After providing $P_1, \ldots, P_n$ with the auxiliary data necessary for the evaluation of $C_f$, $P_h$ resets itself, if it will be employed in future executions of the MPC protocol. Otherwise it shuts down. In the online phase $P_h$ does not take part, it especially neither provides input nor receives output with respect to the evaluation of $C_f$.

Our approach implies, that the helper party $P_h$ sends the generated shares to each party $P_i$, $i = 1, \ldots, n$, privately. Communication between pairs $P_i$ and $P_j$, $1 \leq i < j \leq n$, is not intended. This contrasts traditional approaches, in which the offline phase requires communication among all parties. We therefore reduce the number of necessary secure private channels from $n(n-1)$ between each pair of parties to $n$ channels between $P_h$ and each $P_i$, $i = 1, \ldots, n$. This further improves the efficiency of the offline phase. We point out that this does not have any impact on the online phase, which is explicitly left unmodified in any MPC protocol our approach is applied to.

Depending on the concrete instantiation of $P_h$, the parties $P_1, \ldots, P_n$ must assume, that $P_h$ does not collude with any other party. We give examples for the non-collusion of $P_h$ being a necessary assumption as well as is not being necessary in Section 5.

While $P_h$ has neither input nor output in the online phase, malformed preprocessing data can falsify the output or even reveal a party's input. We thus present $P_h$ in two flavours, depending on whether $P_h$ is trusted by $P_1, \ldots, P_n$, which is indicated by the probability $p_f$.

- If $P_h$ is regarded as a trusted party by $P_1, \ldots, P_n$, we have $p_f = 1$. $P_h$ produces a data set $D'$ corresponding to the distribution of the set $D$. That is for each $D_i$, $i = 1, \ldots, m$, $P_h$ samples a set $D'_i$ of size $\#D_i$ from $X_i$. $P_h$ shares each $d \in D'_i$ among $P_1, \ldots, P_n$ via $\mathcal{S}.\texttt{share}(d)$, where $\mathcal{S}$ is the secret sharing scheme underlying the MPC protocol. The parties $P_1, \ldots, P_n$ utilise the shares received from $P_h$ in place of the output of the offline phase and evaluate $C_f$ accordingly. The online phase hence remains unchanged.
- An untrusted helper party is indicated by $p_f < 1$. The parties $P_1, \ldots, P_n$ agreed on a probability $p_f$, that they accept as a chance of undetected dishonest behaviour on $P_h$'s side. For each $D_i$ in (3.1), $P_h$ generates

$$k_i = \left\lceil \frac{(1 - p_f)}{p_f} \#D_i \right\rceil$$

  additional items according to the distribution of $D_i$ for each $i \in \{1, \ldots, m\}$. We denote the resulting set by $D'_i$, where $\#D'_i = \#D_i + k_i$.

  Prior to evaluating $C_f$ in the online phase, $P_1, \ldots, P_n$ apply a cut-and-choose approach to the shares they received from $P_h$ in that they randomly choose a portion of $k_i$ elements of each set $D'_i$, $i = 1, \ldots, m$, and publicly reconstruct them. If a malformed data item is opened this way, $P_h$ will be considered dishonest and the received data is discarded. Otherwise the data received from $P_h$ is deemed correct and the evaluation of $C_f$ is continued with the remaining unopened data. Since we have $\#D'_i = \#D_i + k_i$ items for each $i = 1, \ldots, m$, a sufficient number of unopened data items remains. We show in Theorem 3, that $p_f$ bounds the probability with which $P_h$ can successfully provide malformed data items without being detected.

$\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{A})$

$c \leftarrow\!\$\ \{0,1\}$
$S' \leftarrow\!\$\ 2^S \setminus \Gamma$
$p \leftarrow\!\$\ [0,1]$
$\mathsf{prep}_0 \leftarrow \mathcal{O}_{\text{real prep}}(C_f, p, S')$
$\mathsf{prep}_1^* \leftarrow P_h(C_f, p)$
$\mathsf{prep}_1 \leftarrow \mathsf{prep}_1^*|_{S'}$
$c' \leftarrow \mathcal{A}(\mathsf{prep}_c)$
**return** $c == c'$

**Fig. 2.** Experiment $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{A})$

$\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\mathcal{A})$

$c \leftarrow\!\$\ \{0,1\}$
$S' \leftarrow\!\$\ 2^S \setminus \Gamma$
**for** $P_i \in S'$
  $x_i \leftarrow\!\$\ X$
  $y_i \leftarrow\!\$\ Y$
**endfor**
$t_0 \leftarrow \mathsf{Sim}_{\mathcal{P}}\left((x_i, y_i)_{P_i \in S'}\right)$
$t_1 \leftarrow \mathsf{Sim}_{\mathcal{P}'}\left((x_i, y_i)_{P_i \in S'}\right)$
$c' \leftarrow \mathcal{A}(t_c)$
**return** $c == c'$

**Fig. 3.** Experiment $\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\mathcal{A})$

*Remark 1.* The scenario of a trusted helper party can be regarded as a special case of the untrusted helper party, where we have $p_f = 1$ and thereby $k_i = 0$.

**Theorem 1 (Cheating probability of $P_h$).** *The probability that $P_h$ generates a malformed data set without detection is upper bounded by $p_f$.*

We give a proof of Theorem 3 in the Appendix.

### 3.1 Security

We will now prove, that the introduction of the helper party does not weaken the security of the MPC protocol. To this end, we show that a simulatable MPC protocol remains simulatable after the introduction of $P_h$.

We model the indistinguishability of the output of a real offline phase from the output of $P_h$ in Experiment $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{A})$ given in Figure 2. In this game, we denote by $\mathcal{O}_{\text{real prep}}(\cdot)$ an oracle, that upon being handed the circuit description $C_f$, the failure probability $p$ and an unauthorised set $S'$ internally executes a preprocessing phase according to $C_f$ and $p$ and outputs the shares of the parties in $S'$.

**Definition 2.** *For a circuit $C_f$, let $\mathcal{A}$ be an arbitrary algorithm. The advantage of $\mathcal{A}$ in Experiment $\mathsf{Exp}^{ind\text{-}prep}_{C_f}(\mathcal{A})$ is defined as*

$$\mathsf{Adv}^{ind\text{-}prep}_{C_f}(\mathcal{A}) = \left| \frac{1}{2} - \Pr\left[\mathsf{Exp}^{ind\text{-}prep}_{C_f}(\mathcal{A}) = \mathsf{true}\right] \right|.$$

**Lemma 1.** *For any unauthorised subset of shareholders $S' \subset \{P_1, \ldots, P_n\}$, the data produced in the preprocessing phase is perfectly indistinguishable from the data provided by $P_h$. That is, for any adversary $\mathcal{A}$ and any circuit $C_f$ and any failure probability $p_f$, we have*

$$\mathsf{Adv}_{C_f}^{ind\text{-}prep}(\mathcal{A}) = 0.$$

We give a proof of this Lemma in the Appendix. We now prove that the MPC protocol, that arises from introducing the helper party to a simulatable MPC protocol, is simulatable itself. We capture this notion in $\mathsf{Exp}^{\text{ind-trans}}(\cdot)$.

**Definition 3.** *The advantage of an adversary $\mathcal{A}$ in Experiment $\mathsf{Exp}_{C_f}^{ind\text{-}trans}(\mathcal{A})$ is defined as*

$$\mathsf{Adv}_{C_f}^{ind\text{-}trans}(\mathcal{A}) = \left| \frac{1}{2} - \Pr\left[\mathsf{Exp}_{C_f}^{ind\text{-}trans}(\mathcal{A}) = \mathsf{true}\right] \right|.$$

*An MPC protocol is* simulatable*, if for any circuit $C_f$ and any adversary $\mathcal{A}$ we have*

$$\mathsf{Adv}_{C_f}^{ind\text{-}trans}(\mathcal{A}) = 0.$$

**Theorem 2.** *Let $\mathcal{P}$ be a simulatable, secret sharing based MPC protocol. And let $\mathcal{P}'$ be an identical protocol, yet with the modification detailed above applied to it, that is the offline phase is executed by a helper party $P_h$. Then $\mathcal{P}'$ is simulatable.*

We give a proof of Theorem 2 in the Appendix.

## 4 SPDZ application and performance

### 4.1 Application to SPDZ

We demonstrate the practicality of our approach by applying it to the offline phase in the SPDZ multi-party computation protocol established by Damgard et al. [8]. SPDZ enables a set of at least two parties to evaluate arithmetic circuits in $\mathbb{Z}_p$ for a prime $p$, where all gate inputs and outputs are certified with a global key. SPDZ thus naturally integrates a correctness measure for the computation.

The offline phase in SPDZ can be considered as a process in two steps. First a public key $\mathsf{pk}$ is established with an according secret key $\alpha$. Each party $P_i$, $1 \leq i \leq n$, obtains a share of $\alpha$. Second the data necessary for the evaluation of each gate in the circuit to be evaluated in the online phase is generated. In SPDZ, we distinguish two methods of sharing. For a value $x$, we denote

$$[x] = \left( (x_1, \ldots, x_n), \left( \beta_i, \gamma(x)_1^i, \ldots, \gamma(x)_n^i \right)_{i=1,\ldots,n} \right),$$

where $\sum_{i=1}^{n} x_i = x$ and $\sum_{i=1}^{n} \gamma(x)_j^i = x\beta_j$ for all $j = 1, \ldots, n$. Each party $P_i$ holds shares $x_i, \beta_i, \gamma(x)_1^i, \ldots, \gamma(x)_n^i$. The secret key $\alpha$ is shared as $[\alpha]$. The sharing $\langle x \rangle$ denotes

$$\langle x \rangle = (\delta, (x_1, \ldots, x_n), (\gamma(x)_1, \ldots, \gamma(x)_n)),$$

where $\delta$ is publicly known and each party $P_i$, $1 \leq i \leq n$, obtains $x_i$ and $\gamma(x)_i$ so that $\sum_{i=1}^{n} x_i = x$ and $\sum_{i=1}^{n} \gamma(x)_i = \alpha(x + \delta)$ hold.

A circuit suitable for SPDZ contains four types of gate: input gate, addition gate, multiplication gate and output gate. The preprocessing for an input gate consists of a random value $r$, that is shared as $\langle r \rangle$ as well as $[r]$. To input a value $x$, a party $P_i$ has the other parties open $[r]$ to him and computes and publishes $\epsilon \leftarrow x - r$. The parties then derive their local shares of $x = \epsilon + \langle r \rangle$. For an addition of two shared values $\langle x \rangle$ and $\langle y \rangle$, the parties locally add their respective shares according to the sharing described above. Thus no preprocessing is required for addition gates. A multiplication gate requires two Beaver triples $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$ and $\langle x \rangle$, $\langle y \rangle$, $\langle z \rangle$, where $ab = c$ and $xy = z$, and a random $[t]$. The sharings $\langle x \rangle$, $\langle y \rangle$, $\langle z \rangle$ and $[t]$ are opened to verify the correctness of the triple $\langle a \rangle$, $\langle b \rangle$ and $\langle c \rangle$, which is then used for the multiplication itself. The preprocessing for an output gate entails a simple shared random value $[r]$.

In assisting the offline phase of SPDZ, the helper party analyses the circuit to be evaluated for the gates types it contains and determines the data to be generated. The helper party $P_h$ then samples the respective data items and shares them among $P_1, \ldots, P_n$ in the appropriate format of $\langle \cdot \rangle$ or $[\cdot]$. We assume the existence of secure private channels between $P_h$ and each $P_i$, $i = 1, \ldots, n$. The discussion on how to instantiate such a channel is out of the scope of this work, we refer to [12] as an example for how instantiate such channels. If $p_f < 1$, that is the helper party is not assumed trusted by $P_1, \ldots, P_n$, $P_h$ samples and shares additional data items according to the value of $p_f$.

### 4.2 Performance

We demonstrate the performance gains of our approach in the offline phase over the original SPDZ protocol as presented in [8] and the improvements proposed in Overdrive [17] and MASCOT [16]. For that we implement the protocol to be executed by $P_h$ in C++. We use the boost library in its version 1.74. This natively enables us to generate and share data items of 128 bits, so that our results are comparable to those of [17,16], who also used 128 bits of randomness in their implementation.

We evaluate the performance of our implementation on commodity PCs in a local network, where the helper party is run on a machine with eight cores and thirty-two GB of RAM. This setup is almost identical to that of the performance test of MASCOT [16].

Our test is executed in two scenarios: First, we measure the maximum possible rate of generating and sharing Beaver triples $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$ and $\langle x \rangle$, $\langle y \rangle$, $\langle z \rangle$ along with the complimenting randomness $[t]$. Second, we execute the offline phase for a Vickrey auction with one hundred bids.

| | This work | | | | SPDZ | MASCOT | | Overdrive | |
|---|---|---|---|---|---|---|---|---|---|
| # of parties | 2 | 3 | 4 | 5 | 2 | 2 | 5 | 2 (low gear) | 2 (high gear) |
| Avg. Beaver tr./s | 134k | 108k | 85k | 69k | 4.2k | 4.8k | 1k | 15k | 2.3k |
| Interquartile range | 3252 | 1521 | 1096 | 741 | no data given | | | | |
| Maximum | 140k | 112k | 87k | 71k | no data given | | | | |
| Minimum | 126k | 103k | 82k | 66k | no data given | | | | |

**Fig. 4.** Benchmark results for 10,000 Beaver triples in $\mathbb{F}_p$, $\lceil \log_2 p \rceil = 128$

**Output of Beaver triples for SPDZ.** We measure the time elapsed for generating the preprocessing for 10,000 multiplication gates, deriving the throughput per second. Our test is executed for two to five parties. We give the resulting performance numbers in Figure 4 with a visual representation in Figure 5. It can be seen, that in the setting of two parties in the online phase, our implementation improves the results of SPDZ and MASCOT at a factor of 30, and those of Overdrive more than nine-fold. In the setting of five parties, our implementation outperforms MASCOT by a factor of 69.

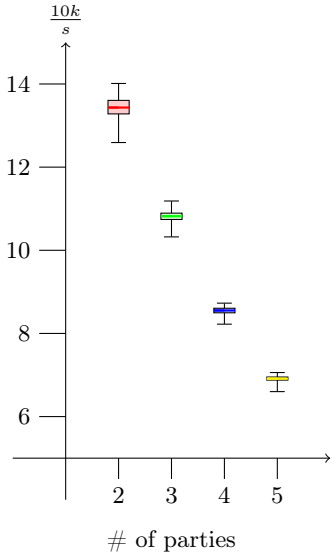


**Fig. 5.** Generating Beaver triples for up to five parties

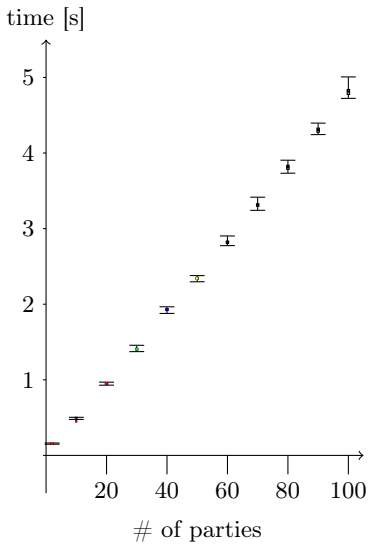


**Fig. 6.** Vickrey auction preprocessing for up to 50 parties

**Vickrey Auction.** In the second scenario, we simulate the preprocessing for a Vickrey auction with one hundred bids. The winner of the auction is the highest bidder, yet the price to be paid by the winning party is the second highest bid. The layout of this auction incentivises the parties to provide realistic bids. A modified version of this is in fact utilised by online auction house eBay.

We simulate the preprocessing for an online phase that is carried out between two up to one hundred parties. We test each setting one hundred times. The circuit uses 44571 Beaver triples. We give the performance numbers in Figure 7.

As can be seen in Figure 6, the time elapsed for the preprocessing increases linearly with the number of parties in the online phase, which agrees with the amount of data to be generated. In MASCOT a setting of two parties was evaluated, which we outperform by a factor of over 60. And in Overdrive the online phase was carried out between 100 parties, which we improved on by a factor of over 20.

| # of parties | 2 | 10 | 20 | 30 | 50 | 60 | 70 | 80 | 90 | 100 | MASCOT | Overdrive |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Avg. time [s] | 0.16 | 0.50 | 0.95 | 1.40 | 2.34 | 2.82 | 3.32 | 3.81 | 4.30 | 4.81 | 10 | 98 |
| Maximum | 0.17 | 0.50 | 0.98 | 1.46 | 2.39 | 2.91 | 3.42 | 3.91 | 4.40 | 5.01 | no data given | |
| Minimum | 0.14 | 0.48 | 0.92 | 1.37 | 2.29 | 2.77 | 3.23 | 3.73 | 4.24 | 4.72 | no data given | |
| IQR | 0.00 | 0.01 | 0.02 | 0.02 | 0.03 | 0.03 | 0.03 | 0.05 | 0.06 | 0.07 | no data given | |

**Fig. 7.** The Vickrey auction preprocessing for up to 50 parties in $\mathbb{F}_p$, $\lceil \log_2 p \rceil = 128$

## 5 Instantiations for $P_h$

In Section 3, we introduced the helper party $P_h$ and proved that a simulatable MPC protocol stays simulatable, if the offline phase is executed by the helper party. We demonstrated the efficiency gained by the introduction of $P_h$ in Section 4.2. In this section we discuss feasible real world instantiations for $P_h$ and their advantages and shortcomings as well as the underlying assumptions. Since the data shared by the helper party is known in plaintext to $P_h$, the role of $P_h$ cannot be assumed by a computing party $P_i$, $1 \leq i \leq n$, itself.

### 5.1 Trusted Execution Environment

The computing parties $P_1, \ldots, P_n$ delegate the offline phase to an agreed upon trusted execution environment (TEE), such as ARM's TrustZone [1] or AMD's Secure Processor [18]. A TEE provides the capability to have an (almost) arbitrary computation executed by an external party in a secure and trusted way. This is achieved by having the TEE prove that the program executed coincides with the program that was given as input via a remote attestation protocol. Thus the parties ascertain, that the TEE executes the protocol for the helper

party and only that. A TEE is to be found in virtually any main stream CPU sold today, thus it is widely available.

To ensure safe communication with the TEE, each party $P_i$, $i = 1, \ldots, n$, establishes a secure and private channel with the TEE by appropriate means. This channel is then used to transmit the shares, that the TEE generates in executing the helper party's task.

Utilising a TEE to implement $P_h$ allows for a entirely counter-intuitive approach: a party $P_j \in \{P_1, \ldots, P_n\}$ that has a TEE at its disposal may provide the other parties access to it and have the helper party's protocol executed in it. With the parties $P_i$, $i \neq j$, establishing private channels to the TEE, $P_j$ cannot obtain knowledge on the shares received by $P_i$, less it breaks the TEE or the channel protocol.

It is reasonable to propose, that the parties forgo the MPC protocol by sending the TEE their private inputs and having it evaluate the circuit $C_f$. Yet modern applications making use of library components such the GNU C library (glibc) or the C mathematical library (libm) require substantial amounts of memory for their execution due to the size of said libraries. In many cases, these exceed the hardware limitations inherent to the TEE implementation, as the 128 MB memory limit of Intel's SGX [2] implementation on Windows operating systems demonstrates. The direct evaluation of large circuits such as privacy preserving machine learning [5,7] hence cannot be achieved in a typical TEE. Executing the offline phase is nevertheless entirely actionable with the amount of data to be persistently held in memory being almost negligible.

Having a TEE implement the helper party allows $P_1, \ldots, P_n$ to assume $P_h$ as trusted. The protocol of $P_h$ can therefore be instantiated in its most efficient configuration.

Overall we claim that a TEE represents a feasible instantiation of $P_h$, since its widespread availability and the helper party protocol being in its most efficient configuration outweigh the limitations inherent to this approach.

### 5.2 Unrelated External Party

The parties $P_1, \ldots, P_n$ may alternatively employ an unrelated external party for the task of $P_h$. This approach distinguishes itself from the former in that $P_h$ is considered untrusted outright. The parties hence agree on a probability $p_f < 1$ as detailed in Section 3 and apply the cut-and-choose method to verify the correctness of the received shares. The external party executing the task of $P_h$ is incentivised to behave honestly by monetary reward. This means, that if the shares are considered honestly generated after $P_1, \ldots, P_n$ verified them, the helper party receives a previously agreed upon payment. With the computational effort of the protocol for the helper party being comparatively low even for large circuits, the monetary reward for the external party is little. As a concrete instantiation the parties may employ a minimalistic cloud instance as can readily be hired at a wide variety of commercial providers.

A major advantage of this approach in comparison to a TEE is that the external party is not limited with respect to the implementation of $P_h$. This results in an efficiently computed preprocessing phase by the external party.

A caveat of this approach is that the parties $P_1, \ldots, P_n$ must assume that $P_h$ is not colluding with either of the parties, as we illustrated in Section 3. Also, the protocol for $P_h$ cannot be instantiated in its most efficient fashion, i.e., a trusted helper party, since $p_f < 1$ must hold.

In fact, the performance statistics presented in Section 4.2 were obtained in this setting, that is on a comodity PC without specialised hardware. Further improved performance can be achieved with the use of hardware specialised for parallel computations.

### 5.3 Minimal Special Purpose Hardware

The parties $P_1, \ldots, P_n$ may deploy a piece of purpose-built computing hardware to execute the task of the helper party $P_h$. The design for said hardware is made public in a format like VHDL, so that any party can verify that the computation carried out agrees with the protocol for $P_h$. As we already discussed in Section 5.1, even a minimalistic hardware design is sufficient for a successful execution of the preprocessing phase. In contrast, purpose built hardware achieves significantly higher efficiency compared to general computing hardware such as a TEE or an external party. Since the design of the purpose-built hardware is public, no trust assumptions have to be placed in the helper party, which enables the protocol of $P_h$ in its most efficient configuration, i.e., $p_f = 1$.

Obtaining a piece of purpose-built hardware is however rather costly compared to the previously proposed instantiations. With the preprocessing data for an MPC protocol being identical with respect to structure and at most distinct in proportion, a piece of dedicated hardware is highly reusable in future executions of the MPC protocol.

The advantages of employing a minimalistic piece of hardware to implement $P_h$ are efficient computation of the preprocessing data and an indepence from a third party hosting a TEE or executing the protocol of $P_h$. In our oppinion these advantages outweigh the caveat of the initial cost of deployment by far.

The real world instantiations of the helper party discussed above are of course not exhaustive. We gave three examples to illustrate, that it is feasible to instantiate $P_h$ without substantial monetary or organisational overhead while obtaining reasonable guarantees with respect to correctness and confidentiality of the generated data.

## 6 Conclusion

In this work we proposed to speed up the offline phase for secret sharing based MPC protocols by introducing a helper party $P_h$ that generates and shares the preprocessing data necessary for the online phase. The resulting MPC protocol

provides the same security guarantees as the MPC protocol without the introduction of $P_h$, in that if the original protocol is simulatable, then so is the protocol after the introduction of $P_h$. We tested the performance of our approach in the setting of an untrusted external party providing the preprocessing for a SPDZ instance on off-the-shelf hardware in a local network setting. Our approach outperforms prior work. Furhermore, we gave three feasible real world instantiations to demonstrate that our approach is not only of a theoretical nature.

## References


1. Arm security technology, building a secure system using trustzone® technology (2009), https://community.arm.com/cfs-file/__key/telligent-evolution-components-attachments/01-2671-00-00-00-00-53-99/PRD29_2D00_GENC_2D00_009492C_5F00_trustzone_5F00_security_5F00_whitepaper.pdf
2. Intel® software guard extensions (intel® sgx), debug and build configurations (2017), https://www.intel.com/content/dam/develop/external/us/en/documents/intel-sgx-build-configuration-737361.pdf
3. Blakley, G.R.: Safeguarding cryptographic keys. Proceedings of AFIPS 1979 National Computer Conference **48**, 313–317 (1979)
4. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. pp. 192–206 (2008). https://doi.org/10.1007/978-3-540-88313-5_13
5. Chaudhari, H., Rachuri, R., Suresh, A.: Trident: Efficient 4PC framework for privacy preserving machine learning (2020)
6. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod $2^k$ for dishonest majority. pp. 769–798 (2018). https://doi.org/10.1007/978-3-319-96881-0_26
7. Damgård, I., Escudero, D., Frederiksen, T.K., Keller, M., Scholl, P., Volgushev, N.: New primitives for actively-secure MPC over rings with applications to private machine learning. pp. 1102–1120 (2019). https://doi.org/10.1109/SP.2019.00078
8. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. pp. 643–662 (2012). https://doi.org/10.1007/978-3-642-32009-5_38
9. Damgård, I., Thorbek, R.: Linear integer secret sharing and distributed exponentiation. pp. 75–90 (2006). https://doi.org/10.1007/11745853_6
10. Demmler, D., Schneider, T., Zohner, M.: ABY - A framework for efficient mixed-protocol secure two-party computation (2015)
11. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation (extended abstract). pp. 554–563 (1994). https://doi.org/10.1145/195058.195408
12. Fischlin, M., Günther, F., Muth, P.: Information-theoretic security of cryptographic channels. pp. 295–311 (2020). https://doi.org/10.1007/978-3-030-61078-4_17
13. Fitzi, M., Garay, J.A., Gollakota, S., Rangan, C.P., Srinathan, K.: Round-optimal and efficient verifiable secret sharing. pp. 329–342 (2006). https://doi.org/10.1007/11681878_17
14. Fujisaki, E., Okamoto, T.: A practical and provably secure scheme for publicly verifiable secret sharing and its applications. pp. 32–46 (1998). https://doi.org/10.1007/BFb0054115

15. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. pp. 218–229 (1987). https://doi.org/10.1145/28395.28420
16. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. pp. 830–842 (2016). https://doi.org/10.1145/2976749.2978357
17. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. pp. 158–189 (2018). https://doi.org/10.1007/978-3-319-78372-7_6
18. Malhotra, A.: Amd ryzen™ pro 5000 series mobile processors, making defenses count: Designing for substantial depth (2021), https://www.amd.com/system/files/documents/amd-security-white-paper.pdf
19. Orsini, E., Smart, N.P., Vercauteren, F.: Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. pp. 254–283 (2020). https://doi.org/10.1007/978-3-030-40186-3_12
20. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. pp. 129–140 (1992). https://doi.org/10.1007/3-540-46766-1_9
21. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). pp. 73–85 (1989). https://doi.org/10.1145/73007.73014
22. Scholl, P., Smart, N.P., Wood, T.: When it's all just too much: Outsourcing MPC-preprocessing. pp. 77–99 (2017)
23. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (Nov 1979)
24. Tassa, T.: Hierarchical threshold secret sharing **20**(2), 237–264 (Apr 2007). https://doi.org/10.1007/s00145-006-0334-8
25. Traverso, G., Demirel, D., Buchmann, J.A.: Dynamic and verifiable hierarchical secret sharing. pp. 24–43 (2016). https://doi.org/10.1007/978-3-319-49175-2_2
26. Yao, A.C.C.: Protocols for secure computations (extended abstract). pp. 160–164 (1982). https://doi.org/10.1109/SFCS.1982.38

## Appendix 1.A Proofs

*Proof (Proof of Lemma 1).* Each data item produced in a real preprocessing phase is information theoretically hidden from $S'$. This holds for each data item, that was generated and shared among the computing parties by $P_h$. Since each data item is sampled independently from each other, the data sets received are perfectly indistinguishable.

*Proof (Proof of Theorem 2).* We prove Theorem 2 in three steps: first we give a simulator for any unauthorised subset of $\{P_1, \ldots, P_n\}$ in the protocol $\mathcal{P}'$, second we prove the indistinguishability of said simulator from that of the original protocol $\mathcal{P}$ which gives us the simulatibility of $\mathcal{P}'$ in the third step.

We denote by $C_f$ the circuit, that $\mathcal{P}$ and hence $\mathcal{P}'$ are to evaluate. Let $S' \notin \Gamma$ be an arbitrary unauthorised subset of $\{P_1, \ldots, P_n\}$ and let $\mathsf{Sim}_{\mathcal{P}}$ denote the simulator of $\mathcal{P}$. Thus upon receiving

$$\left\{(x_i, y_i)_{P_i \in S'}\right\}$$

as input, $\mathsf{Sim}_{\mathcal{P}}$ outputs a transcript that is indistinguishable from $\{\mathsf{view}_i\}_{P_i \in S'}$.

We give a simulator $\mathsf{Sim}_{\mathcal{P}'}$ for $\mathcal{P}'$. The simulator $\mathsf{Sim}_{\mathcal{P}'}$ uses $\mathsf{Sim}_{\mathcal{P}}$ in the following manner. Let $\left\{(x'_i, y'_i)_{P_i \in S'}\right\}$ denote the input of $\mathsf{Sim}_{\mathcal{P}'}$. $\mathsf{Sim}_{\mathcal{P}'}$ runs $\mathsf{Sim}_{\mathcal{P}}\big((x'_i, y'_i)_{P_i \in S'}\big)$ and returns whatever $\mathsf{Sim}_{\mathcal{P}}$ outputs.

It remains to prove, that the output of $\mathsf{Sim}_{\mathcal{P}'}$ indistinguishable from that of $\mathsf{Sim}_{\mathcal{P}}$. We capture this in Experiment $\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\mathcal{A})$ in Figure 3. We give a reduction of the preprocessing distinguishing problem to the simulator distinguishing problem to show the hardness of the former. Hence let $\mathcal{D}$ be an adversary in $\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\cdot)$ with positive advantage. We construct a polynomial-time adversary $\mathcal{D}'$ against $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\cdot)$ that uses $\mathcal{D}$ to gain the same advantage. The input to an adversary $\mathcal{D}'$ in $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{D}')$ is a set of preprocessing shares $\{\mathsf{prep}^*_i\}_{P_i \in S'}$ for an unauthorised set $S'$. To simulate $\mathsf{Exp}^{ind-trans}_{C_f}(\cdot)$ to $\mathcal{D}$, $\mathcal{D}'$ samples $(x_i)_{P_i \in S'}$ and $(y_i)_{P_i \in S'}$ from their respective distributions. $\mathcal{D}'$ then hands

$$t_c \overset{\text{perf}}{=} \mathsf{Sim}_{\mathcal{P}}\big((x_i, y_i)_{P_i \in S'}\big)$$

to $\mathcal{D}$. $\mathcal{D}'$ then outputs whatever decision bit $\mathcal{D}$ outputs. It remains to argue, that $\mathcal{D}'$ has the same advantage in $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{D}')$ as $\mathcal{D}$ has in $\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\mathcal{D})$. The output of a simulator $\mathsf{Sim}\big((x_i, y_i)_{P_i \in S'}\big) = (x_i, (r_i)_j, (m_i)_j)_{P_i \in S'}$ looks identically distributed to an unauthorised set of parties $S' \notin \Gamma$. The randomness $(r_i)_j$ is either shared among all parties or locally sampled for each party $P_i$ in a secret sharing based MPc protocol. Thus for any $\{(r_i)_j\}_{P_i \in S'}$ we have

$$\begin{aligned}\mathsf{Sim}_{\mathcal{P}}\big((x_i, y_i)_{P_i \in S'}\big) = (x_i, (r_i)_j, (m_i)_j)_{P_i \in S'} &\overset{\text{perf}}{=} \\ (x_i, (r'_i)_j, (m_i)_j)_{P_i \in S'} &= \mathsf{Sim}'_{\mathcal{P}}\Big(\big((x_i, y_i), (r'_i)_j\big)_{P_i \in S'}\Big),\end{aligned}$$

where $\mathsf{Sim}'_{\mathcal{P}}$ outputs the same as $\mathsf{Sim}_{\mathcal{P}}$, but replaces the randomness $(r_i)_j$ for $(r'_i)_j$. This gives us

$$\mathsf{Sim}_{\mathcal{P}}\left((x_i, y_i)_{P_i \in S'}\right) \overset{\text{perf}}{=} \mathsf{Sim}_{\mathcal{P}'}\left((x_i, y_i)_{P_i \in S'}\right)$$

and thus

$$\mathsf{Adv}^{\text{ind-prep}}_{\mathcal{D}'}(C_f) = \mathsf{Adv}^{\text{ind-trans}}_{\mathcal{D}}(C_f).$$

With Lemma 1, this gives us $\mathsf{Adv}^{\text{ind-trans}}_{\mathcal{D}'}(C_f) = 0$, thus the output of $\mathsf{Sim}'_{S'}$ is thus indistinguishable from that of $\mathsf{Sim}_{S'}$, which in turn is indistinguishable from the real view $\mathsf{view}_{S'}$. The MPC protocol $\mathcal{P}'$ is therefore simulatable.

*Proof (Proof of Theorem 3).* Fix an arbitrary subset $D'_i$, $1 \leq i \leq m$. Let $a$ denote the number of malformed items in $D'_i$. If $a > \#D_i$, then the computing parties necessarily selects a malformed data item for opening, since $\#D'_i = \#D_i + k_i$. The probability of successful cheating for $P_h$ is thus 0. We hence assume $a \leq \#D_i$. Let us first consider the case of single malformed data item in $D_i$, i.e., $a = 1$. The probability of $P_1, \ldots, P_n$ not selecting the malformed item is therefore

$$\frac{\binom{\#D'_i - 1}{k}\binom{1}{0}}{\binom{\#D'_i}{k}} = \frac{\binom{\#D_i + k - 1}{k}\binom{1}{0}}{\binom{\#D_i + k}{k}} = \frac{\frac{(\#D_i + k - 1)!}{k!(\#D_i - 1)!}}{\frac{(\#D_i + k)!}{k!\#D_i!}}$$
$$= \frac{\#D_i}{\#D_i + k} \leq \frac{\#D_i}{\#D_i + \frac{1 - p_f}{p_f}\#D_i} = p_f.$$

The statement therefore holds for $a = 1$. For $a > 1$, that is more than one malformed data item, the probability of malformed items not being selected and opened in the cut-and-choose paradigm is clearly upper bounded by the probability in the case of $a = 1$. Therefore, $P_h$ can only successfully cheat with a probability at most $p_f$.