Uncovering Algebraic Structures in the MPC Landscape

Navneet Agarwal[1], Sanat Anand[1], and Manoj Prabhakaran*[1]

[1]Indian Institute of Technology Bombay, {navneet,sanat,mp}@cse.iitb.ac.in

Abstract

A fundamental problem in the theory of secure multi-party computation (MPC) is to characterize functions with *more than 2 parties* which admit MPC protocols with information-theoretic security against passive corruption. This question has seen little progress since the work of Chor and Ishai (1996), which demonstrated difficulties in resolving it. In this work, we make significant progress towards resolving this question in the important case of aggregating functionalities, in which m parties $P_1, \ldots, P_m$ hold inputs $x_1, \ldots, x_m$ and an aggregating party P_0 must learn $f(x_1, \ldots, x_m)$.

We uncover a rich class of algebraic structures that are closely related to secure computability, namely, "Commuting Permutations Systems" (CPS) and its variants. We present an extensive set of results relating these algebraic structures among themselves and to MPC, including new protocols, impossibility results and separations. Our results include a necessary algebraic condition and slightly stronger sufficient algebraic condition for a function to admit information-theoretically secure MPC protocols.

We also introduce and study new models of minimally interactive MPC (called UNIMPC and UNIMPC*), which not only help in understanding our positive and negative results better, but also open up new avenues for studying the cryptographic complexity landscape of multi-party functionalities. Our positive results include novel protocols in these models, which may be of independent practical interest.

Finally, we extend our results to a definition that requires UC security as well as semi-honest security (which we term *strong security*). In this model we are able to carry out the characterization of *all* computable functions, except for a gap in the case of aggregating functionalities.

*Supported by the Dept. of Science and Technology, India via the Ramanujan Fellowship and an Indo-Israel Joint Research Project grant, 2018.

Contents

1 Introduction

Secure Multi-Party Computation (MPC) is a central and unifying concept in modern cryptography. The foundations, as well as the applications, of MPC have been built up over a period of almost four decades of active research since the initial ideas emerged [SRA79, Blu81, Yao82]. Yet, some of the basic questions in MPC remain open. Specifically, the following basic problem remains open to this day for various standard notions of security (when there are no restrictions like honest majority):

Which multi-party functions admit information-theoretically secure MPC?

Indeed, one of the most basic forms of this problem remains wide open: for the case of security against passive corruption, a characterization of securely realizable functions is known only for 2-party functions [Kus89]. Chor and Ishai pointed out the difficulty of this problem, by disproving a natural conjecture for characterizing securely realizable k-party functionalities in terms of functionalities involving fewer parties [CI96]. Since then, very little progress has been made on this problem.

In this work, we make significant progress towards resolving this question in the important case of *aggregating functionalities*: In an aggregating functionality, there are m parties $P_1, \ldots, P_m$ with inputs $x_1, \ldots, x_m$ and an aggregating party P_0 must learn $f(x_1, \ldots, x_m)$. Aggregating functionalities form a practically and theoretically important class. In particular, it has been the subject of an influential line of study that started with the *minimal model for secure computation* of Feige, Kilian and Naor [FKN94]. This model – also referred to as the Private Simultaneous Messages (PSM) model [IK97] – served as a precursor of important concepts like randomized encodings [IK00] that have proven useful in a variety of cryptographic applications. Recently, a strengthening of this model, called Non-Interactive MPC (NIMPC) was introduced by Beimel et al. [BGI+14], which is closer to standard MPC in terms of the security requirements.[1] However, these models do not address the question of secure realizability in the standard model, because due to weakened security requirements, all aggregating functions are securely realizable in these models.

Towards characterizing secure realizability under (the standard model of) MPC, we uncover and examine a rich class of algebraic structures of aggregating functionalities. We exploit these structures to give new positive and negative results for MPC. Further, we also put forth new minimalistic, yet natural models of secure computation that arise from these results. These new models and algebraic structures, in tandem, open up new avenues for investigating the landscape of secure multiparty computation involving many parties.

Commuting Permutations Systems. We identify an algebraic-combinatorial structure called Commuting Permutations System (CPS) and interesting sub-classes thereof. CPS generalizes the function of abelian group summation to a less structured class of functions. Indeed, as a function of two inputs (denoted as $m = 2$), a CPS can be identified with a *quasigroup* operation, or equivalently the function specified by a minor of a Latin square. (For $m > 2$ inputs, CPS imputes more structure than m-dimensional Latin hypercubes.)

[1] Both PSM and NIMPC consider protocols of the following form: a coordinator sends a private message to each of $P_1, \ldots, P_m$; each P_i uses this message and its input to compute a single message which it sends to P_0; P_0 computes an output. PSM has a corruption model in which only P_0 could be corrupted, whereas NIMPC allows any subset of the parties (other than the coordinator) to be corrupted. But when such corruption takes place, NIMPC allows the adversary to learn the *residual function* determined by the honest parties' inputs – i.e., the output for each possible setting of the inputs for the corrupt parties (unlike in MPC, where the output for only a given input of the corrupt parties is learned).

We define **CPS** as the class of all aggregating functions which *embed* into a CPS functionality (Definition 2). We also identify two interesting sub-classes of CPS that (as we shall see) are closely related to secure computability, corresponding to Commuting Permutation *Subgroup* Systems (CPSS) and *Complete* CPS (CCPS).

Minimal Models of MPC. In a parallel thread, we develop new minimalistic models of MPC, that help us study feasibility of information-theoretic MPC. These models (called UNIMPC* and UNIMPC) admit secure protocols only for functions which have secure protocols in the standard MPC model. We remark that ours is perhaps the first significant minimalistic model with this property, as previous minimalistic models – PSM [FKN94] and NIMPC [BGI+14] – admit secure protocols for all functions.

UNIMPC stands for *Unassisted NIMPC* and, as the name suggests, removes the assistance from the trusted party in NIMPC: Instead the parties should securely compute the correlated randomness by themselves, in an offline phase. Unlike PSM and NIMPC, which have an incorruptible party, *UNIMPC retains the standard security model of MPC*, allowing corruption of any set of parties, and requiring the adversary to learn nothing more than the output of the function.

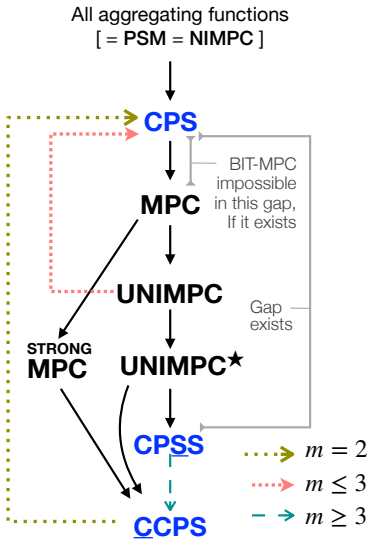

Figure 1: The m-PC landscape of aggregating functions. The classes in blue typeface are defined in terms of algebraic/combinatorial properties, and the others in terms of secure computability. Arrow $\mathbf{A} \to \mathbf{B}$ indicates $\mathbf{A} \supseteq \mathbf{B}$.

> *A UNIMPC protocol is an MPC protocol and can also be immediately interpreted as an NIMPC protocol.*[2]

Note that MPC and NIMPC are incomparable in the sense that an MPC protocol does not yield an NIMPC protocol (because of the general communication pattern) and an NIMPC protocol does not yield an MPC protocol (because of the use of a trusted party, and because the adversary is allowed to learn potentially more than the output of the function). Thus UNIMPC could be seen as a common denominator of these two secure computation models.

UNIMPC* corresponds to a minimalistic version of UNIMPC, with protocols which have a single round of (simultaneous) communication among the parties before they get their inputs, followed by a single message from each party to the aggregator after they receive their input. (UNIMPC allows arbitrarily many rounds of communication prior to receiving inputs.)

Strongly Secure MPC. We also study feasibility under a *stronger* model of MPC, which requires both UC security and passive security to hold simultaneously (information theoretically). Traditionally, UC security refers to the setting of active corruption, in which the security guarantees are relative to an ideal model where too the corrupt parties are actively corrupt. While stronger in general, this gives a weaker guarantee than security against passive corruption, when the corrupt

[2] Replacing the views from the pre-processing phase of a UNIMPC protocol with correlated randomness from a trusted party turns it into an NIMPC protocol.

parties are indeed only passively corrupt.[3] From a practical point of view, strong security (possibly weakened to hold only against PPT adversaries) is important, and arguably the "right" notion in many cases. Here we initiate the study of characterizing multi-party functionalities that are strongly securely realizable.

Relating Secure Computation to the Algebraic Classes. Our results show the rich connections between the cryptographic complexity landscape of MPC and the combinatorial/algebraic structures of the functions, as summarized in Figure 1. We briefly point out the several results that go into making this map. All results relate to the information-theoretic setting with finite functions.

- ☐ **MPC** $\subseteq$ **CPS**: This result hinges on characterizing the following cryptographic property algebraically: given any subset of the inputs and the output of the function, the *residual function* of the remaining inputs can be determined. (Theorem 2.)
- ☐ **CPSS** $\subseteq$ **UNIMPC***: We establish this by developing a novel MPC protocol that generalizes the simple abelian group summation protocol to a certain class of (non-abelian) group actions (Theorem 3).
- ☐ **CPSS** $\subsetneq$ **CPS**: We give a concrete family of functions that fall into the gap between these two classes (Theorem 1). Combined with the above results, this separation leaves an intriguing gap between the necessary and sufficient conditions for MPC. (But we show in Theorem 4, that this gap disappears/reduces for a small number of input parties.)
- ☐ **CCPS** $\subseteq$ **UNIMPC***: The class **CCPS** (for Complete CPS) consists of the "Latin Hypercube" functionalities that fall within **CPS**. We show that all such functions, in more than two dimensions, are highly structured and in particular fall within **CPSS** (Lemma 6). For two dimensions, i.e., Latin squares, this is not true; but in this case a UNIMPC* protocol can be directly given for all Latin squares. Further, in this case, due to a classical result of Ryser [Rys51], **CPS** = **CCPS** (see Section 1.3).
- ☐ **UC security results**: The characterization of UC securely realizable functions has been resolved for *2 and 3-party functionalities* [CKL06, PR08a], but remains open for more than 3 parties. Prabhakaran and Rosulek [PR08a] showed that there are only two classes of secure function evaluation functionalities – aggregating and disseminating – that can possibly have UC secure protocols. They also gave a UC secure protocol for the "disseminated OR" functionality for 3 parties. We build on this further to show that:
  - Disseminated OR functionality with any number of players is UC securely realizable. Further, every disseminating functionality is UC securely realizable by a reduction to the disseminated OR functionality (Section 8.2).
  - Every aggregating functionality in **CCPS** has a UC secure protocol; this relies on a compiler from a strongly secure protocol for $\mathcal{F}$ (which exists only if $\mathcal{F}$ is a CPS functionality) to one for $\mathcal{F}$ restricted to a domain D (Section 8.1).
  - In both these positive results, we obtain strong security (Theorem 7). Combined with the negative results (Theorem 6), this shows that

$$\textbf{CCPS} \cup \textbf{DISS} \subseteq \textsc{strong}\textbf{MPC} \subseteq \textbf{CPS} \cup \textbf{DISS}$$

[3] E.g., a 2-party functionality in which Bob receives $a \vee b$, where $a, b \in \{0, 1\}$ are inputs to Alice and Bob respectively, has no protocol secure against passive corruption; but a protocol in which Alice simply sends a to Bob is UC secure. Also see $\mathcal{F}_{\text{AND}}$ discussed in Section 8.1.

where **STRONGMPC** denotes the class of *all* functionalities (not just aggregating functionalities) that have strongly secure protocols, and **DISS** and **CCPS** are interpreted as all functionalities "isomorphic" to functionalities that are disseminating or functionalities that embed into a CCPS functionality. In Figure 1, this relationship is indicated restricted to aggregating functionalities (in which, case the extension to isomorphism – which allows all parties to have inputs and outputs – can be ignored).

□ **Additional Results and Implications:**

• Recently, Halevi et al. introduced the notion of "Best Possible Information-Theoretic MPC" (BIT-MPC) [HIKR18], by removing the trusted party and the non-interactive structure in the NIMPC model, but retaining the provision that (in the ideal-world) the adversary is allowed to learn the residual function of the honest parties' inputs. While the set of functions for which BIT-MPC is possible is a strict superset of **MPC**, the main open problem posed in [HIKR18] is whether all functions have BIT-MPC protocols. We note that for all functions in **CPS**, BIT-MPC protocols are automatically MPC protocols (because for them the residual function can be deduced from the output and the corrupt parties' own inputs). Thus if $\mathbf{CPS} \setminus \mathbf{MPC} \neq \emptyset$, then there exist functions which do not have a BIT-MPC protocol.

• Our necessity result – that $\mathbf{MPC} \subseteq \mathbf{CPS}$ – can be extended in a couple of ways (Section 5.1): Firstly, the necessity condition continues to hold even if the corruption model allowed the corruption of at most one party other than the aggregating party, if we require a UNIMPC protocol (this model could be called 1-Robust UNIMPC).

Secondly, the necessity of being contained in **CPS** holds even for NIMPC (even 1-Robust NIMPC), if we required an additional security property that the adversary learns only what the output and its own inputs reveal (like in MPC), rather than the residual function of the honest parties (as in NIMPC or BIT-MPC). On the other hand, for functions in **CPS**, the protocols in the original NIMPC model remain secure in the new model too. Thus for this model, the set of realizable functions is exactly **CPS**.

• While our focus is on aggregating functionalities, our positive results for passive-secure MPC do yield new protocols for *symmetric functionalities* wherein all parties get the same output – as considered in [CI96]. This is because a passive-secure MPC protocol for an aggregating functionality can be readily converted into one for a symmetric functionality computing the same function.

• Since one of our results (Theorem 4) depends on the existence of NIMPC protocols, we present a simple NIMPC protocol for general functionalities in Appendix A. This protocol is a generalization of an NIMPC protocol in [HIJ+16] to arbitrary input domains, presented more directly in terms of the function matrix. This NIMPC protocol is more efficient and much simpler than the earlier ones in the literature [BGI+14, OY16].

We present more details of our results and techniques in Section 1.3.

1.1 Open Problems

We leave several open problems, which relate to understanding MPC as a whole, and various aspects of it individually. While, by definition, $\mathbf{MPC} \supseteq \mathbf{UNIMPC} \supseteq \mathbf{UNIMPC}^{\star}$, it is open to show if these inclusions are strict. Their exact relations with the combinatorial classes **CPS**, **CPSS** and **CCPS** also remain open.

CPS is an exact characterization of functions realizable in a model obtained by allowing UNIMPC protocols to use a trusted party to generate the views in the offline phase (this model being the same as the variant of NIMPC mentioned aboved, with the extra security property that the adversary learns only what the output and its own inputs reveal). Therefore, separating **UNIMPC** from **CPS** requires a better understanding of multiparty secure sampling [PP12], and to separate **UNIMPC** and **UNIMPC***, we need a better understand the power of interaction for secure sampling.

Interestingly, we leave open a combinatorial problem as well: Is **CCPS** $\subsetneq$ **CPSS** for $m > 2$? (For $m = 2$, we have a containment in the opposite direction, since **CCPS** = **CPS** $\supsetneq$ **CPSS**.) This corresponds to the question of whether every CPSS can be embedded into a CCPS.

In Appendix C, we give a few explicit functions in **CPS** for which we do not have an MPC, UNIMPC or UNIMPC* protocol, and leave them as open challenges.

1.2 Related Work

There has been a large body of work aimed at characterizing functionalities with MPC protocols in various models (see, e.g., a survey [MPR13]). For some important classes, exact characterizations are known: this includes passive and active (stand-alone) security for 2-party deterministic functions [Kus89, KMR09, MPR09], multi-party functions with restricted adversary structures [BGW88, CCD88, HM97], multi-party functions with binary alphabet [CK91], multi-party protocols which only have public communication [KMR09], and UC security for 2-party functions [CKL06, PR08a].

The characterization question for the multi-party setting (with point-to-point channels and no honest majority, for passive security) was explicitly considered in [CI96]. It was shown there that there exist m-party functions which do not have any passive-secure protocol such that the $m-1$-party function obtained by merging any two parties results in a securely realizable functionality. This problem in the context of UC security was studied in [PR08a], where the terms aggregating functionality and disseminating functionality were coined.

The NIMPC model was introduced by Beimel et al. [BGI$^+$14], inspired by the earlier work of Feige et al. [FKN94]. This was generalized to other patterns of interaction in [HIJ$^+$16]. A computational version of UNIMPC (but with a public-key infrastructure) was recently explored in [HIJ$^+$17].

A recent independent and concurrent work by Halevi et al. [HIKR18] overlaps with some of our results. Specifically, they also observe the fact that an MPC protocol must reveal the residual function of the honest parties to an adversary corrupting the output party, which is the staring point of our proof of Theorem 2 (they do not derive the combinatorial characterization of CPS). The transformation from NIMPC to UNIMPC we use to prove Theorem 4 is a special case of the NIMPC to MPC compiler of [HIKR18], which forms the main tool for their positive results. Finally, as pointed out above, the main open problem left in [HIKR18] is whether there are functions with no BIT-MPC protocol, and this relates to an open problem we leave, namely whether **CPS** = **MPC**: A negative answer to our question answers that of [HIKR18] in the negative.

1.3 Technical Overview

We give a brief overview of CPS functions, and a couple of our protocols that exploit this structure.

An $m+1$ aggregating functionality involves parties $P_1, \cdots, P_m$ with inputs and an aggregator P_0 who learns the output. A classical example of an aggregating functionality that admits secure

computation is the summation operation in an abelian group. As a starting point to understanding all securely computable functions, one could try to generalize this function. Consider the 3-party version of this problem, involving two input parties P_1, P_2 and an output party P_0. W.l.o.g. we can consider computing a function $f : [n_1] \times [n_2] \to [n]$, given an as a matrix M with $M_{ij} = f(i, j)$. Suppose there is a passive secure protocol Π for computing f. From the results on 2-party MPC we know that an adversary which passively corrupts $\{P_0, P_1\}$ must learn P_2's input fully (up to equivalent inputs). Then, for this protocol to be secure, even given an ideal functionality, an adversary who passively corrupts $\{P_0, P_1\}$ should be able to learn P_2's input. A passive adversary is not allowed to change the parties' inputs. Hence, for any inputs $x_1 \in [n_1], x_2 \in [n_2]$, it must be the case that $(x_1, f(x_1, x_2))$ uniquely determines x_2. Symmetrically, $(x_2, f(x_1, x_2))$ uniquely determines x_1. We refer to this as the *Latin property* of M, named after Latin squares. (Latin squares are $n \times n$ square matrices in which each row and each column is a permutation of $[n]$. Note that a square matrix with the Latin property is the same as a Latin square.)

It is easy to see that any 3-party aggregating functionality $f : [n] \times [n] \to [n]$ which is a Latin square has a passive secure protocol: P_1 and P_2 privately agree on a random permutation σ over $[n]$, and then P_1 sends P_0 the row indexed by its input x_1, but with positions permuted according to σ: i.e., a vector $(z_1, \cdots, z_n)$ where $z_{\sigma(j)} = M_{x_1,j}$. P_2 sends $k = \sigma(x_2)$ to P_0, and P_0 outputs $z_k = M_{x_1,x_2}$. Note that the security of this protocol relies on not only the Latin property, but also on the fact that each row has all n elements. However, since any rectangle with the Latin property can be embedded into an (at most quadratically larger) Latin square [Rys51], any function f which has the Latin property does indeed have a passive secure protocol.

This might suggest that for arbitrary number of parties, an analogous Latin hypercube property would be a tight characterization of secure computability. Interestingly, this is not the case. With m input clients, the 2-party results imply that an adversary corrupting a subset of the m input parties and the aggregator P_0 can learn the *residual function* of the honest parties' inputs. Since the passive adversary cannot change the input of the corrupt parties even in the ideal world, this means that any choice of the corrupt parties' inputs should reveal the residual function of the honest parties. We identify an algebraic formulation in terms of a "Commuting Permutation System" (CPS) that captures this condition tightly.

A CPS over the output alphabet $[n]$ has input sets $X_i \subseteq S_n$, for $i = 1$ to m, where S_n is the group of all permutations of $[n]$. On input $(\pi_1, \cdots, \pi_m) \in X_1 \times \cdots \times X_m$, the output is defined as $\pi_1 \circ \cdots \circ \pi_m(1)$. The "commuting" property is the requirement that this output is invariant to the order in which the m permutations are applied to 1. Note that the commutativity needs to hold only when applied to 1. Also, it holds only *across* the sets $X_1, \cdots, X_m$. That is if $\pi, \pi' \in X_i$, it is not necessary that $\pi \circ \pi'(1)$ equals $\pi' \circ \pi(1)$. The function table of a CPS functionality is indeed a Latin hypercube, but the converse does not hold.

Being a CPS functionality is necessary to have an MPC protocol (let alone a UNIMPC protocol). Unfortunately, we do not know if this is also a sufficient condition. But given some additional structure in a CPS, we are able to give a new protocol. The additional structure that we can exploit is that each X_i is a *subgroup* of S_n, in which case we call the system a Commuting Permutation *Subgroups* System or CPSS. Exploiting this property, we design a protocol for computing CPSS functions, as discussed below.

UNIMPC Protocol for CPSS Functionalities. We present a novel protocol with perfect, information-theoretic security against passive corruption for all CPSS functionalities (and, further, is in fact, UC secure for a sub-class). Recall that the goal is to let P_0 learn $\pi_1 \circ \cdots \circ \pi_m(1)$, where π_i is

a permutation that P_i receives as input. At first glance, our protocol may appear similar in structure to a protocol for an abelian group sum: each party P_i shares its input π_i as $\pi_i = \sigma_{i,0} \circ \sigma_{i,1} \circ \cdots \circ \sigma_{i,m}$, where each of the shares itself belongs to X_i. It will be helpful to visualize these shares as forming the i^{th} row in a matrix of shares. The shares in each column $(\sigma_{1,j}, \cdots, \sigma_{m,j})$ for $j \in [m]$ will be correlated with each other in some manner, so that the output can be reconstructed by aggregating only the shares $(\sigma_{1,0}, \cdots, \sigma_{m,0})$. (An analogy for the case of the abelian group would be to choose the shares in each column to sum up to the identity element.) These shares will be sent to P_0.

But there are a couple of major differences. Firstly, permutations do not commute in general, and it is not clear how the shares can be meaningfully combined. Secondly, we *must not reveal* the composition of the inputs – i.e., the permutation $\pi_1 \circ \cdots \circ \pi_m$ – to the aggregator; only the result of applying this composition to 1 should be revealed. So, choosing the column shares to "add up to" the identity permutation would be problematic, not to mention that there may not be any such choice other than choosing all the shares to be the identity element.

In our protocol, we choose the column shares such that their composition has 1 as a fixed point (there is at least one such choice, since the each entry can be chosen as the identity permutation). Then, using the CPSS property, it can be shown that $(\prod_{i\in[m]} \sigma_{i,0})(1) = (\prod_{i\in[m]} \pi_i)(1)$ (see Figure 2). It turns out that we can use the subgroup structure in CPSS to argue that if the shares are chosen uniformly at random subject to the above constraint, then $(\sigma_{1,0}, \cdots, \sigma_{m,0})$ reveals nothing more than $\pi_1 \circ \cdots \circ \pi_m(1)$.

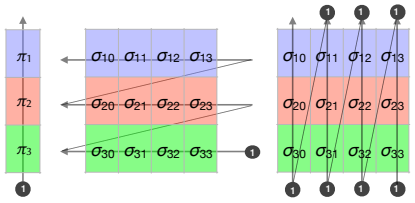

Figure 2: Elements in the i^{th} row belong to a subgroup X_i in a CPSS. The subgroup structure enables secret-sharing as $\pi_i = \prod_{j=m}^{0} \sigma_{i,j}$. Then the illustrated quantities are equal: $\left(\prod_{i\in[m]} \pi_i\right)(1) = \left(\prod_{i\in[m]} \prod_{j=m}^{0} \sigma_{i,j}\right)(1) = \left(\prod_{j=m}^{0} \prod_{i\in[m]} \sigma_{i,j}\right)(1)$. The last equality relies on the closure property in the subgroup, as well as the commutativity guarantee (when applied to 1). In our protocol, for each $j > 0$, $\left(\prod_{i\in[m]} \sigma_{i,j}\right)(1) = 1$, and hence this also equals $\left(\prod_{i\in[m]} \sigma_{i,0}\right)(1)$.

Further, even if we consider all the shares $\sigma_{i,j}$ *except for* $(i,j) \in S \times S$ for some $S \subseteq [m]$, we show that they reveal nothing more than the residual function $\left(\prod_{i\in S} \pi_i\right)(1)$. The need to consider revealing this set of shares comes from the fact that our protocol is not an NIMPC protocol (where a trusted dealer could compute $\sigma_{i,j}$ for all $(i,j) \in [m]^2$ and send only $(\sigma_{i,1}, \cdots, \sigma_{i,m})$ to each party P_i); instead we require the parties to compute all the shares themselves, which is achieved by each party P_j computing the j^{th} column of shares, and distributing it among all the parties P_i. Thus when we consider a set S of honest parties, only the shares $\sigma_{i,j}$ where $(i,j) \in S^2$ remain hidden from the adversary.

UC-secure Protocols. It turns out that the above protocol for aggregating functions is UC secure if the function is a Complete CPSS (CCPSS) function. For $m \geq 3$, a Complete CPS is always a Complete CPSS, and hence this gives a UC secure (in fact, strongly secure) protocol for all CCPS functionalities. (The case of $m = 2$ is handled separately.)

However, for a function that is only *embedded in* a CCPS functionality, this protocol is not necessarily UC secure (because nothing prevents an adversary from using an input from the full domain of the CCPS functionality). We give a compiler that can take a UC secure protocol for a CCPS functionality, and transform it into a UC secure protocol for the functionality restricted to a smaller domain. The main idea of the compiler is to run several instances of the original protocol

with the parties using random inputs from the restricted domain. That they used inputs from the restricted domain is then verified using a cut-and-choose phase. Then, an aggregated AND functionality is used to identify instances among the unopened executions to obtain the output. Plugging in a simple UC secure protocol for aggregated AND, this compiler yields a UC secure protocol. Interestingly, though aggregated AND itself has no strongly secure protocol (or passive-secure protocol, for that matter) as it is not a CPS functionality, the resulting protocol above is a strongly secure protocol.

We remark that this is a feasibility result that relies on the domains being finite (small) as the compiler's overhead is polynomial in the domain size.

We also present a reduction from any disseminating function to the disseminated-OR functionality. This is also a feasibility result that relies on the number of parties being finite (small) as the protocol is exponential in the number of parties. To complete establishing the realizability of all disseminating functions, we give a UC secure protocol for the disseminated-OR functionality (extending a 3-party protocol for the same functionality in [PR08a]).

2 Preliminaries

We write $[n]$ to denote the set $\{1, \cdots, n\}$. S_n denotes the symmetric group over $[n]$, namely, the group of all permutations of $[n]$. In our proofs, we shall use the product notation $\prod$ to denote the composition operation of permutations. Note that composition of permutations is a non-commutative operation in general, and hence the order of the indices is important (as in $\prod_{i=1}^{t} \rho_i$). When the order is not important, we denote the indices by a set (as in $\prod_{i \in [t]} \rho_i$).

Below we define notions referred to through out the paper. Additional notions relevant to strong security are deferred to Section 8.

We adapt the definition of an *aggregating functionality* from [PR08a].[4]

Definition 1 (Aggregating Functionality)**.** *An $(m+1)$ party* Aggregating functionality *accepts inputs $x_i \in X_i$ from P_i for $i = 1$ to m, and sends $f(x_1, \cdots, x_m)$ to party P_0, where $f : X_1 \times \cdots \times X_m \to \Omega$ is a fixed function.*

Consistent with the literature on feasibility questions, we consider the functions to have constant-sized domains (rather than infinite domains or domains expanding with the security parameter). Also, in all our positive results, the security obtained is perfect and hence the protocols themselves do not depend on the security parameter. Our negative results do allow protocols to have a negligible statistical error in security.

Definition 2 (Embedding)**.** *An aggregating functionality $f : X_1 \times \cdots \times X_m \to [n]$ is said to* embed *into a functionality $g : X'_1 \times \cdots \times X'_m \to [n']$ if there exist functions $\phi_i : X_i \to X'_i$ for $i \in [m]$, and an injective function $\phi_0 : [n] \to [n']$ such that for all $(x_1, \cdots, x_m) \in X_1 \times \cdots \times X_m$,*

$$\phi_0(f(x_1, \cdots, x_n)) = g(\phi_1(x_1), \cdots, \phi_m(x_m)). \tag{1}$$

Below, $\mathcal{A} \cong \mathcal{B}$ denotes that the statistical difference between the two distributions $\mathcal{A}$ and $\mathcal{B}$ is negligible as a function of a (statistical) security parameter.

[4] We allow only the aggregating party P_0 to have an output. The original definition in [PR08a] allows all the parties to have outputs, but requires that for each party other than P_0, its output is a function only of its own input. Such a function is "isomorphic" to an aggregated functionality as we define here.

Definition 3 (Passive Secure MPC). *An $(m+1)$-party protocol Π with parties $P_1, \cdots, P_m, P_0$ is said to be an information-theoretically secure MPC protocol for an $(m+1)$-party aggregating functionality f against passive corruption, if for any subset $T \subseteq [m] \cup \{0\}$, there exists a simulator S s.t. for any input $x \in X$:*

$$\text{VIEW}_{\Pi(x)}(\{P_i | i \in T\}) \cong \begin{cases} S(x_T, f(x)) & \text{if } 0 \in T \\ S(x_T, \perp) & \text{otherwise} \end{cases}$$

where $\text{VIEW}_{\Pi(x)}(\{P_i | i \in T\})$ represents the view of the parties $\{P_i | i \in T\}$ in an execution of Π with input x and $\perp$ represents an empty input.

We shall use the following result for 2-party MPC, obtained from the general characterization in [KMR09].

Lemma 1 (2-Party MPC with one-sided output [KMR09]). *If a finite 2-party functionality which takes inputs $x \in X$ and $y \in Y$ from Alice and Bob respectively and outputs $f(x, y)$ to Bob for some function $f : X \times Y \to Z$ has a statistically secure protocol against passive adversaries, then $\forall x, x' \in X$ it holds that $\exists y \in Y, f(x, y) = f(x', y) \Rightarrow \forall y \in Y, f(x, y) = f(x', y)$.*

Residual Function. For a domain $X = X_1 \times \cdots \times X_m$, and a set $T \subseteq [m]$, we write X_T to denote $X_{i_1} \times \cdots \times X_{i_t}$, where $T = \{i_1, \cdots, i_t\}$ (in sorted order). For ease of description, we index the coordinates of an element $x \in X_T$ by the elements of T. Then, given $f : X \to Y$ and $x^* \in X_{\overline{T}}$, we define the residual function $f_{x^*} : X_T \to Y$ as $f_{x^*} : x \mapsto f(z)$ where $z_i = x^*_i$ if $i \in \overline{T}$ and x_i if $i \in T$.

NIMPC Protocol. Below we summarize the definition of an NIMPC protocol [BGI+14].

Definition 4 (NIMPC: Syntax and Correctness). *Let $X_1, \cdots, X_m, R_1, \cdots, R_m, M_1, \cdots, M_m$ and Ω be finite domains. Let $X = X_1 \times \cdots \times X_m$ and let $\mathcal{F}$ be a family of functions $f : X \to \Omega$. A non-interactive secure multiparty computation (NIMPC) protocol for $\mathcal{F}$ is a triplet $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ where*

- $\text{Gen} : \mathcal{F} \to R_1 \times \cdots \times R_m$ *is a randomized function.*
- Enc *is an m-tuple of deterministic functions $(\text{Enc}_1, \cdots, \text{Enc}_m)$, where $\text{Enc}_i : X_i \times R_i \to M_i$,*
- Dec*: $M_1 \times \cdots \times M_m \to \Omega$ is a deterministic function satisfying the following correctness requirement: for any $x = (x_1, \cdots, x_m) \in X$ and $f \in \mathcal{F}$,*

$$Pr[r = (r_1, \cdots, r_m) \leftarrow \text{Gen}(f) : \text{Dec}(\text{Enc}(x, r)) = f(x)] = 1,$$

where $\text{Enc}(x, r) = (\text{Enc}_1(x_1, r_1), \cdots, \text{Enc}_m(x_m, r_m))$.

The communication complexity of Π is the maximum of $\log |R_1|, \cdots, \log |R_m|, \log |M_1|, \cdots, \log |M_m|$.

Definition 5 (NIMPC Security). *We say that an NIMPC protocol Π for $f : X \to \Omega$ is T-robust for $T \subseteq [m]$, if there exists a randomized function Sim (a "simulator") such that $\forall x^* \in X_{\overline{T}}$, we have $\text{Sim}(f_{x^*})$ distributed dentically as $(M_{\overline{T}}, R_T)$, where R and M are the joint randomness and messages defined by $R \leftarrow \text{Gen}(f)$ and $M_i \leftarrow \text{Enc}_i(x_i, R_i)$, where $x \in X$ is such that $x_{\overline{T}} = x^*$. Π is said to be* secure *if it is T-robust for all $T \subseteq [m]$.*

A Private Simultaneous Message (PSM) protocol [FKN94] is simply a $\emptyset$-robust NIMPC.

UC Security. Finally, we shall also use the definition of UC security [Can05]. Briefly, the definition compares a "real world" execution of a protocol Π for a functionality $\mathcal{F}$, with an "ideal world" execution where $\mathcal{F}$ is implemented using a trusted party, instead of the protocol. For simplicity, we shall consider static corruption. Then, every adversary $\mathcal{A}$ in the real world or $\mathcal{S}$ in the ideal world is allowed to corrupt a fixed set of parties. Writing $\text{REAL}(\mathcal{Z}, \mathcal{A}, \Pi)$ and $\text{IDEAL}(\mathcal{Z}, \mathcal{S}, \mathcal{F})$ for the distribution of the environment's output in these two executions, Π is said to be a UC-secure protocol for $\mathcal{F}$ if

$$\forall \mathcal{A}\ \exists \mathcal{S}\ \forall \mathcal{Z},\ \text{IDEAL}(\mathcal{Z}, \mathcal{S}, \mathcal{F}) \cong \text{REAL}(\mathcal{Z}, \mathcal{A}, \Pi),\ \text{and } \mathcal{S} \text{ corrupts same set as } \mathcal{A}.$$

3 New Models

In this section we define UNIMPC and UNIMPC*, which are models of secure computation, as well as combinatorial objects CPS and CPSS. For simplicity, we define UNIMPC and UNIMPC* for fixed functions rather than function families (though the definitions can be easily extended to function families, where all the input players receive the function as an input).

Definition 6 (UNIMPC)**.** *We define an* Unassisted Non-Interactive Secure Multi-party Computation (UNIMPC) protocol Π *for an* $(m+1)$*-party aggregating functionality* $f : X \to \Omega$ *as* $\Pi = (\mathcal{R}, \text{Enc}, \text{Dec})$ *where:*

- $\mathcal{R}$ *is an* m*-party randomized protocol (without inputs), generating correlated views* $(r_1, \cdots, r_m) \in R_1 \times \cdots \times R_m$*.*
- Enc *is an* m*-tuple of deterministic functions* $(\text{Enc}_1, \cdots, \text{Enc}_m)$ *where* $\text{Enc}_i : X_i \times R_i \to M_i$*.*
- $\text{Dec} : M_1 \times \cdots \times M_m \to \Omega$ *is a deterministic function satisfying the following correctness requirement: for any* $(x_1, \cdots, x_m) \in X$ *and any view* $(r_1, \cdots, r_m)$ *which* $\mathcal{R}$ *generates with positive probability,*

$$\text{Dec}((\text{Enc}_1(x_1, r_1), \cdots, \text{Enc}_m(x_m, r_m)) = f(x_1, \cdots, x_m).$$

It is identified with a two-phase MPC protocol where:

1. **Offline Phase:** *The parties* $P_i : i \in [m]$ *run* $\mathcal{R}$ *(without any input) so that each* P_i *obtains the view* r_i*.*
2. **Online Phase:** *Every* P_i *encodes its input* x_i *as* $z_i = \text{Enc}_i(x_i, r_i)$ *and sends it to the aggregator* P_0*.* P_0 *outputs* $\text{Dec}(z_1, \cdots, z_m)$*.*

Security: A UNIMPC protocol Π *for* $f : X \to \Omega$ *is said to be* T-secure *(for* $T \subseteq [m]$*) if there exists a simulator* S *s.t. for any* $x \in X$*:*

$$\text{VIEW}_{\Pi(x)}(\{P_i | i \in T\} \cup \{P_0\}) \cong S(x_T, f(x))$$

where $\text{VIEW}_{\Pi(x)}(\cdot)$ *represents the view of a given set of parties in the two-phase protocol above, with input* x*.*

For any $t \in [m]$*,* Π *is said to be* t-robust *if it is* T*-secure* $\forall T \subseteq [m]$ *s.t.* $|T| \leq t$*. A UNIMPC protocol* Π *is said to be* secure *if it is* m*-robust.*

We point out that a secure UNIMPC protocol as defined above is a passive secure MPC protocol for f (as in Definition 3). Note that in defining T-security we considered only the case when the set of corrupt parties includes the aggregator. But when the aggregator is honest, security is automatically guaranteed by the structure of the UNIMPC protocol (the view of the adversary being derived completely from the offline phase).

Definition 7 (UNIMPC*)**.** *We define an* Unassisted Non-Interactive Secure Multi-party Computation protocol with Non-Interactive Pre-Processing *(UNIMPC* protocol)* Π *for a functionality* $f : X \to \Omega$ *as a UNIMPC protocol* $\Pi = (\mathcal{R}, \text{Enc}, \text{Dec})$ *for* f *where* $\mathcal{R}$ *consists of a single round (i.e., each party simply sends messages to the others, and then receives all the messages sent to it).*

We define classes **MPC**, **UNIMPC**, **UNIMPC*** as the class of aggregating functionalities that have (information-theoretically) passive secure MPC, UNIMPC and UNIMPC* protocols, respectively.

4 Commuting Permutations System

In this section, we define the new algebraic-combinatorial classes.

Definition 8 (CPS and CPSS)**.** *An* (n, m)-Commuting Permutations System *(CPS) is a collection* $(X_1, \cdots, X_m)$ *where for all* $i \in [m]$, $X_i \subseteq S_n$ *contains the identity permutation, and for any collection* $(\pi_1, \cdots .\pi_m)$ *with* $\pi_i \in X_i$, *and* $\rho \in S_m$, $\pi_1 \circ \cdots \circ \pi_m(1) = \pi_{\rho(1)} \circ \cdots \circ \pi_{\rho(m)}(1)$.[5]

It is called an (n, m)-Commuting Permutation Subgroups System *(CPSS) if each* X_i *is a subgroup of* S_n.

Note that given a CPS $(X_1, \cdots, X_m)$, for any $(\pi_1, \cdots, \pi_m) \in X_1 \times \cdots \times X_m$, the expression $(\prod_{i\in[m]} \pi_i)(1)$ is well-defined as the order of composition is not important.

Definition 9 (CCPS)**.** *An* (n, m)*-CPS* $(X_1, \cdots, X_m)$ *is said to be* complete in dimension i *if* $\{\pi(1) \mid \pi \in X_i\} = [n]$. *If it is complete in all* m *dimensions, it is called a* Complete CPS *(CCPS).*

Definition 10. *An* $(m + 1)$*-party aggregating functionality* $f : X_1 \times \cdots \times X_m \to [n]$ *is said to be a* CPS functionality *(resp.,* CPSS and CCPS functionality*) if* $(X_1, \cdots, X_m)$ *is an* (n, m)*-CPS (resp.,* (n, m)*-CPSS and* (n, m)*-CCPS), and for all* $(\pi_1, \cdots, \pi_m) \in X_1 \times \cdots \times X_m$, $f(\pi_1, \cdots, \pi_m) = (\prod_{i\in[m]} \pi_i)(1)$.

CPS *(resp.,* **CPSS** *and* **CCPS***) is defined as the class of all aggregating functionalities that embed into a CPS functionality (resp., CPSS functionality and CCPS functionality).*

A CPSS enjoys a certain (non-abelian) group structure. More specifically, the CPSS $(G_1, \cdots, G_m)$ can be identified with a group, with the set of elements $G_1 \times \cdots \times G_m$ and group operation $*$ defined as $(\sigma_1, \ldots, \sigma_m) * (\sigma'_1, \ldots, \sigma'_m) = (\sigma_1 \circ \sigma'_1, \ldots, \sigma_m \circ \sigma'_m)$. This is captured in the following lemma.

Lemma 2. *Suppose* $(G_1, \cdots, G_m)$ *is a CPSS. Then, for any set of* mt *permutations* $\{\sigma_{i,j} \mid i \in [m], j \in [t]\}$ *such that* $\sigma_{i,j} \in G_i$, *it holds that*

$$(\prod_{j=1}^{t} \prod_{i\in[m]} \sigma_{i,j})(1) = (\prod_{i\in[m]} \prod_{j=1}^{t} \sigma_{i,j})(1).$$

[5] Choice of 1 is arbitrary. Requiring identity permutation to always be part of each X_i is w.l.o.g., as a CPS without it will remain a CPS on adding it.

Proof. Consider $\rho \circ \prod_{i=1}^{m} \rho_i(1)$, where $\rho_i \in G_i$ for each i, and $\rho \in G_{i_0}$ for some $i_0 \in [m]$. Note that the order of composition is not important in $\prod_{i=1}^{m} \rho_i(1)$, since $(G_1, \cdots, G_m)$ is a CPS(S), and we may write it as $\prod_{i\in[m]} \rho_i(1)$. Also, define ρ'_i as

$$\rho'_i = \begin{cases} \rho \circ \rho_{i_0} & \text{if } i = i_0 \\ \rho_i & \text{otherwise.} \end{cases}$$

Since G_{i_0} is a group, we have $\rho'_i \in G_i$ for all $i \in [m]$ (including i_0). Then,

$$(\rho \circ \prod_{i\in[m]} \rho_i)(1) = (\rho \circ \rho_{i_0} \circ \prod_{i\in[m]\setminus\{i_0\}} \rho_i)(1) = (\rho'_{i_0} \circ \prod_{i\in[m]\setminus\{i_0\}} \rho'_i)(1) = (\prod_{i\in[m]} \rho'_i)(1)$$

where in the last step, we again used the CPS property. The claim follows by repeatedly using the above equality. □

The following lemma regarding embedding a CPS into a CPSS will be useful in our first result.

Lemma 3. *If an (n, m)-CPS $(X_1, \cdots, X_m)$ embeds into an (n', m)-CPSS $(G_1, \cdots, G_m)$, then it embeds into an (n', m)-CPSS $(G'_1, \cdots, G'_m)$ and there exist functions $\phi_i : X_i \to G'_i$ for $i \in [m]$, such that ϕ_i maps the identity permutation over $[m]$ to itself, and for all $(\sigma_1, \cdots, \sigma_m) \in X_1 \times \cdots \times X_m$,*

$$\prod_{i\in[m]} \sigma_i(1) = \prod_{i\in[m]} \phi_i(\sigma_i)(1). \tag{2}$$

Proof. Note that Equation 2 is the same as Equation 1 from Definition 2, except for omitting ϕ_0 and specializing to the case that the functionalities f and g are CPS functionalities. Hence, compared to Definition 2, the above statement makes two more requirements: Firstly, $\phi_i(\pi_0) = \pi_0$ for each $i \in [m]$, where π_0 denotes the identity permutation over $[m]$; secondly, ϕ_0 is the identity function. We show that an embedding which may not satisfy these conditions can be transformed to one which does.

To enforce the first condition, we transform ϕ_i to $\hat{\phi}_i$ defined as follows. Suppose each $\phi_i(\pi_0) = \sigma_i$. Then, we define $\hat{\phi}_i(\pi) = \sigma_i^{-1} \circ \phi_i(\pi)$ (so that $\hat{\phi}_i(\pi_0) = \pi_0$), and $\hat{\phi}_0 = (\sigma_1 \circ \cdots \circ \sigma_m)^{-1} \circ \phi_0$. We claim that this is a valid embedding from $(X_1, \cdots, X_m)$ to $(G_1, \cdots, G_m)$. To see this, firstly note that $\hat{\phi}_i$ indeed maps elements of X_i into elements of G_i, because G_i is a group. Secondly, from Lemma 2, it follows that in a CPSS with $\{\sigma_i, \beta_i\} \subseteq G_i$,

$$(\prod_{i\in[m]} (\sigma_i^{-1} \circ \beta_i))(1) = (\prod_{i\in[m]} \sigma_i^{-1}) \circ (\prod_{i\in[m]} \beta_i)(1).$$

Then we verify that if $(\phi_0, \phi_1, \cdots, \phi_m)$ satisfies Equation 1, so does $(\hat{\phi}_0, \hat{\phi}_1, \cdots, \hat{\phi}_m)$: For any $(\pi_1, \cdots, \pi_m) \in X_1 \times \cdots \times X_m$,

$$\prod_{i\in[m]} \hat{\phi}_i(\pi_i)(1) = (\prod_{i\in[m]} \sigma_i^{-1}) \circ (\prod_{i\in[m]} \phi_i(\pi_i))(1) = (\prod_{i\in[m]} \sigma_i^{-1}) \circ \phi_0 \circ (\prod_{i\in[m]} \pi_i)(1) = \hat{\phi}_0(\prod_{i\in[m]} \pi_i(1)).$$

Now suppose we have an embedding $\phi_0, \phi_1, \cdots, \phi_m$ where $\phi_i(\pi_0) = \pi_0$. We shall transform it into an embedding $(\hat{\phi}_0, \hat{\phi}_1, \cdots, \hat{\phi}_m)$, into a related CPSS where $\hat{\phi}_0$ is the identity function. To define the new CPSS, consider the homomorphism (in fact, automorphism) $\theta : S_{n'} \to S_{n'}$ given by

$\theta(\alpha) = \phi_0^{-1} \circ \alpha \circ \phi_0$. The new CPSS we define is $(G'_1, \cdots, G'_m)$, where $G'_i = \{\theta(\alpha) | \alpha \in G_i\}$. Note that since θ is a homomorphism, like G_i, G'_i is also a subgroup of $S_{n'}$. Further, for $\beta_i \in G'_i$ where $\beta_i = \theta(\alpha_i)$, we have $\prod_{i=1}^m \beta_i(1) = \phi_0^{-1} \circ (\prod_{i\in[m]} \alpha_i) \circ \phi_0(1)$, showing that it is a CPS. We define the new embedding by setting $\hat{\phi}_i(\pi) = \theta(\phi_i(\pi))$, and $\hat{\phi}_0$ to be the identity function π_0. To see that this is an embedding, we note that $\prod_{i\in[m]} \hat{\phi}_i(\pi_i)(1) = \phi_0^{-1} \circ \prod_{i\in[m]} \phi_i(\pi_i) \circ \phi_0(1)$. But we note that $\phi_0(1) = 1$: this is implied by Equation 1, by considering $x_1 = \cdots = x_m = \pi_0$, and noting that we already have $\phi_i(\pi_0) = \pi_0$. Hence, the new embedding satisfies Equation 1 if the original one does:

$$\hat{\phi}_0^{-1} \circ \prod_{i\in[m]} \hat{\phi}_i(\pi_i)(1) = \prod_{i\in[m]} \hat{\phi}_i(\pi_i)(1) = \phi_0^{-1} \circ \prod_{i\in[m]} \phi_i(\pi_i) \circ \phi_0(1) = \phi_0^{-1} \circ \prod_{i\in[m]} \phi_i(\pi_i)(1).$$

□

Our first result is a separation:

Theorem 1. CPSS $\subsetneq$ CPS.

Proof. We prove this by giving an explicit (n, m)-CPS $(X_1, \cdots, X_m)$ for every value of $m \geq 2$ (and $n = 2^{m-1} + 1$), and showing that the corresponding CPS functionality does not embed into any (n', m)-CPSS functionality. (Also, in Appendix C, we give an instance of an $(n, 2)$-CPS that cannot be embedded into a CPSS.)

As output alphabet we shall use $G \cup \{\perp\}$, where G is an abelian group with the following structure (0 denotes the identity of the group, and summation refers to the group operation):

- $\exists v_1, \cdots, v_m \in G$ such that $\sum_{i=1}^m v_i = 0$, but for every non-empty $T \subsetneq [m]$, $\sum_{i\in T} v_i \neq 0$.
- For all $v \in G$, $v + v = 0$.

Concretely, we may use $G = \{0, 1\}^{m-1}$ (with coordinate-wise XOR being the group operation) and define $v_1, \cdots, v_m$ as follows: For $i = 1$ to $m - 1$, v_i has a single 1 at position i, and $v_m = 1^{m-1}$.

We identify $0 \in G$ with 1 in the output alphabet. Then, the CPS $(X_1, \cdots, X_m)$ has $X_i = \{\pi_0, \pi_i\}$, where π_0 is the identity permutation over $G \cup \{\perp\}$ and π_i is defined as follows:

$$\pi_i(x) = \begin{cases} 0 & \text{if } x = \perp \\ \perp & \text{if } x = v_i \\ x + v_i & \text{otherwise.} \end{cases}$$

It can be easily verified that this is a CPS: for any $T \subsetneq [m]$, $\prod_{i\in T} \pi_i(0) = \sum_{i\in T} v_i$, and $\prod_{i\in[m]} \pi_i(0) = \perp$, with the order of application of the permutations being immaterial. Below, we shall argue that this CPS does not embed into any (n', m)-CPSS.

Suppose, for some n', there is an (n', m)-CPSS, $(G_1, \cdots, G_m)$, and functions $\phi_i : X_i \to G_i$, as specified in Lemma 3. Let $\phi_i(\pi_i) = \rho_i$. Then, for any $T \subseteq [m]$, applying Equation 2 to $\sigma_i = \pi_i$ for $i \in T$ and $\sigma_i = \pi_0$ for $i \notin T$, we can write

$$\prod_{i\in T} \pi_i(0) = \prod_{i\in T} \rho_i(0). \tag{3}$$

Now, to derive a contradiction, we note that since $(G_1, \cdots, G_m)$ is a CPSS, $\rho_1^2 \in G_1$. Then, we require that $\rho_1^2 \circ \rho_2(0) = \rho_2 \circ \rho_1^2(0)$. But,

$$\begin{aligned}
\rho_1^2 \circ \rho_2(0) &= \rho_1(\rho_1 \circ \rho_2(0)) \overset{(a)}{=} \rho_1(\pi_1 \circ \pi_2(0)) \overset{(b)}{=} \rho_1(v_1 + v_2) \overset{(c)}{=} \rho_1(v_3 + \cdots + v_m) \\
&\overset{(b)}{=} \rho_1(\prod_{i=3}^{m} \pi_i(0)) \overset{(a)}{=} \rho_1(\prod_{i=3}^{m} \rho_i(0)) \overset{(a)}{=} \pi_1(\prod_{i=3}^{m} \pi_i(0)) \overset{(b)}{=} v_1 + v_3 + \cdots v_m \overset{(c)}{=} v_2 \\
\rho_2 \circ \rho_1^2(0) &\overset{(a)}{=} \rho_2 \circ \rho_1(\pi_1(0)) \overset{(b)}{=} \rho_2 \circ \rho_1(v_1) \overset{(c)}{=} \rho_2 \circ \rho_1(v_2 + \cdots + v_m) \\
&\overset{(b)}{=} \rho_2 \circ \rho_1 \circ \prod_{i=2}^{m} \pi_i(0) \overset{(a)}{=} \rho_2 \circ \rho_1 \circ \prod_{i=2}^{m} \rho_i(0) \overset{(a)}{=} \rho_2 \circ \pi_1 \circ \prod_{i=2}^{m} \pi_i(0) \overset{(b)}{=} \rho_2(\perp).
\end{aligned}$$

where equalities (a) follow from Equation 3, (b) from the definition of π_i, and (c) from the fact that $\sum_{i=1}^{m} v_i = 0$. Hence we require $\rho_2(\perp) = v_2$. On the other hand, we also have $\rho_2(0) = \pi_2(0) = v_2$, yielding a contradiction, as ρ_i are permutations. □

5 Only CPS Functionalities have (UNI)MPC Protocols

We show that if an aggregating functionality has a statistically secure MPC protocol against semi-honest adversaries (without honest majority or setups), then it must be a CPS functionality. Since UNIMPC protocols are MPC protocols, this applies to UNIMPC as well.

Theorem 2. *If an aggregating functionality has an information-theoretically secure MPC protocol against semi-honest adversaries, then it embeds into a CPS functionality.*

Proof. Suppose an $(m+1)$-party aggregating functionality $f : X_1 \times \cdots \times X_m \to [n]$ is semi-honest securely realizable. Denote the aggregating party as P_0 and for each $i \in [m]$, the party with input domain X_i as P_i.

Firstly, w.l.o.g., we may assume that no party has two *equivalent inputs*, by considering an embedding if necessary. Further, we may let $X_i = [n_i]$ for each i, and $f(1, \cdots, 1) = 1$, by relabeling the inputs and the outputs.

Now, for each $i \in [m]$, consider the 2-party SFE functionality obtained by grouping parties $\{P_j | j \in [m] \setminus \{i\}\}$ as a single party Alice, and the parties $\{P_i, P_0\}$ as a single party Bob. This functionality has the form in Lemma 1, namely, only Bob has any output. Then applying the lemma, we get the following (where the notation $\mathbf{x}[i : \ell]$ denotes the vector obtained from $\mathbf{x}$ by setting x_i to ℓ): $\forall \mathbf{x}, \mathbf{x}' \in X_1 \times \cdots \times X_m$,

$$f(\mathbf{x}) = f(\mathbf{x}') \text{ and } x_i = x_i' \;\Rightarrow\; \forall \ell \in X_i, f(\mathbf{x}[i : \ell]) = f(\mathbf{x}'[i : \ell]). \tag{4}$$

We use this to prove the following claim.

Claim 1. *For each $i \in [m]$ and $\ell \in X_i$, there exists a permutation $\pi_\ell^{(i)}$ such that, for all $\mathbf{x} \in X_1 \times \cdots \times X_m$ with $x_i = 1$,*

$$\pi_\ell^{(i)}(f(\mathbf{x})) = f(\mathbf{x}[i : \ell]). \tag{5}$$

Proof. Fix $i \in [m]$, $\ell \in X_i$. Now, consider defining a (partial) function $\pi_\ell^{(i)}$ using Equation 5. This is well-defined thanks to Equation 4: Even though there could be multiple $\mathbf{x}$ with $x_i = 1$ and the same value for $f(\mathbf{x})$, Equation 4 ensures that they all lead to the same value for $f(\mathbf{x}[i : \ell])$.

Further, with this definition, if $\pi_\ell^{(i)}(a) = \pi_\ell^{(i)}(b)$, this means that there exist $\mathbf{x}, \mathbf{x}'$ with $x_i = x'_i = 1$, $f(\mathbf{x}) = a$, $f(\mathbf{x}') = b$ and $f(\mathbf{x}[i : \ell]) = f(\mathbf{x}'[i : \ell])$. But by considering $\mathbf{z} = \mathbf{x}[i : \ell]$, $\mathbf{z}' = \mathbf{x}'[i : \ell]$, we have $z_i = z'_i$ and $f(\mathbf{z}) = f(\mathbf{z}')$. Hence, by Equation 4, we have $f(\mathbf{z}[i : 1]) = f(\mathbf{z}'[i : 1]$. But since $\mathbf{x} = \mathbf{z}[i : 1]$ and $\mathbf{x}' = \mathbf{z}'[i : 1]$, this means that $a = f(\mathbf{x}) = f(\mathbf{x}') = b$. Hence, $\pi_\ell^{(i)}$ is a one-to-one function, from $\{a | \exists \mathbf{x}, x_i = 1, f(\mathbf{x}) = a\} \subseteq [n]$ to $[n]$. We can arbitrarily extend this to be a permutation over $[n]$ to meet the condition in the claim. □

Finally, for any $\mathbf{x}$ such that $x_{i_1} = \cdots = x_{i_t} = 1$, and distinct $i_1, \cdots, i_t$, by iteratively applying Equation 5, $\pi_{\ell_t}^{(i_t)} \circ \cdots \circ \pi_{\ell_1}^{(i_1)}(f(\mathbf{x})) = f(\mathbf{x}[i_1 : \ell_1] \cdots [i_t : \ell_t])$. Taking $(i_k, \ell_k) = (\rho(k), z_{\rho(k)})$ for any permutation $\rho \in S_m$ and any $\mathbf{z} \in X_1 \times \cdots \times X_m$, we have $\mathbf{x}[i_1 : \ell_1] \cdots [i_m : \ell_m] = \mathbf{z}$, for any $\mathbf{x}$. Then, with $\mathbf{x} = (1, \cdots, 1)$ we get that

$$f(\mathbf{z}) = \pi_{z_{\rho(1)}}^{(\rho(1))} \circ \cdots \circ \pi_{z_{\rho(m)}}^{(\rho(m))}(1),$$

where we substituted $f(\mathbf{x}) = 1$. This concludes the proof that f embeds into the CPS functionality with input domains $\hat{X}_i = \{\pi_\ell^{(i)} | \ell \in [n_i]\}$. □

5.1 Extensions to 1-Robust UNIMPC and NIMPC

Since every secure UNIMPC protocol is a secure MPC protocol, Theorem 2 applies to UNIMPC as well. But it extends to UNIMPC in a stronger manner than it holds for MPC. Note that if we restrict the number of corrupt parties to be at most $m/2$, then every $m+1$ party functionality has a passive secure MPC protocol, even if the functionality is a non-CPS aggregating functionality. But we show that as long as the adversary can corrupt just two parties (the aggregator and one of the input parties), the only aggregating functionalities that have secure UNIMPC protocols are CPS functionalities.

To see this, we consider how Equation 4 was derived in the proof of Theorem 2 (the rest of the argument did not rely on the protocol). We used the given $(m+1)$-party protocol to derive a secure 2-party protocol to which Lemma 1 was applied. In arguing that this 2-party protocol is secure we considered two corruption patterns in the original protocol: the adversary could corrupt $\{P_0, P_i\}$ (Bob) or $\{P_j \mid j \in [m] \setminus \{i\}\}$ (Alice). Now, if we allow only corruption of up to two parties, we cannot in general argue that the resulting two party protocol is secure when Alice is corrupted. However, if the starting protocol was a UNIMPC protocol, then in the resulting 2-phase protocol, there is an offline phase when Alice and Bob interact without using their inputs, and after that Alice sends a single message to Bob in the second phase. *Any such protocol* is secure against the corruption of Alice, as Alice's view can be perfectly simulated without Bob's input. Thus, when the starting protocol is a UNIMPC protocol that is T-secure for every T of the form $\{0, i\}$ $(i \in [m])$, then Lemma 1 applies to the 2-party protocol constructed, and the rest of the proof goes through unchanged. Thus, an aggregating functionality f has a 1-robust UNIMPC protocol only if it is a CPS functionality.

The above argument extends in a way to 1-robust NIMPC as well. Of course, every function has a secure NIMPC protocol [BGI+14], and we cannot require all such functions to be CPS. But we note that NIMPC turned out to be possible for all functions not only because a trusted party is allowed (to generate correlated randomness), but also because NIMPC allows the adversary (corrupting the aggregator and some set of parties) to learn the residual function of the honest parties' inputs. So, one may ask for which functionalities does the adversary *learn nothing more than the output of the*

function on any input (just as in the security requirement for MPC), even as we allow a trusted party to generate correlated randomness. Here, we note that the above argument in fact extends to the NIMPC setting with the trusted party: We simply include the trusted party as part of Alice in the above 2-party protocol. Since the security of the 2-party protocol relied only on security against Bob (and the 2-phase nature of the protocol), including the trusted party as part of Alice does not affect our proof. Thus we conclude that only CPS functionalities have 1-robust NIMPC where the simulator takes only the input of the corrupt parties and the output of the function (rather than the residual function of the honest parties' inputs).

6 UNIMPC Protocols

In this section we present our positive results for UNIMPC* and UNIMPC (Theorem 3 and Theorem 4).

Theorem 3. *Any function embeddable in a CPSS function has a UNIMPC* protocol with perfect security.*

To prove Theorem 3 it is enough to present a perfectly secure protocol for a CPSS function: the protocol retains security against passive corruption when the input domains are restricted to subsets.

UNIMPC* Protocol for CPSS Function.

For $i \in [m]$, party P_i has input $\pi_i \in G_i$, where $(G_1, \cdots, G_m)$ is an (n, m)-CPSS. Party P_0 will output $\pi_1 \circ \cdots \circ \pi_m(1)$.

1. **Randomness Computation:** For each $j \in [m]$, P_j samples $(\sigma_{1j}, \cdots, \sigma_{mj})$ uniformly at random from $G_1 \times \cdots \times G_m$, conditioned on
$$\sigma_{1j} \circ \sigma_{2j} \circ \cdots \circ \sigma_{mj}(1) = 1. \tag{6}$$
For each $i, j \in [m]$, P_j sends σ_{ij} to P_i.
2. **Input Encoding:** P_i computes $\sigma_{i0} := \pi_i \circ (\sigma_{i1} \circ \cdots \circ \sigma_{im})^{-1}$, and sends it to P_0. Note that $(\sigma_{i0}, \cdots, \sigma_{im})$ is an additive secret-sharing of π_i in the group G_i.
3. **Output Decoding:** P_0 outputs $\sigma_{1,0} \circ \sigma_{2,0} \circ \cdots \circ \sigma_{m,0}(1)$.

By construction, the protocol has the structure of a UNIMPC* protocol. Indeed, it is particularly simple for a UNIMPC* protocol in that the randomness computation protocol in offline phase is a single round protocol. Below we argue that this protocol is indeed a perfectly secure protocol for computing $\left(\prod_{i\in[m]} \pi_i\right)(1)$ against passive corruption of any subset of parties.

Perfect Correctness: The output of P_0 is $\prod_{i=0}^m \sigma_{i,0}(1)$. By Equation 6 (applied to $j = 1$) we may write $1 = \prod_{i=1}^m \sigma_{i1}(1)$. We further expand 1 in this expression again by applying Equation 6 successively for $j = 2, \cdots, m$ to obtain $1 = \prod_{j=1}^m \prod_{i=1}^m \sigma_{ij}(1)$. Hence, the output of P_0 may be written as $\prod_{j=0}^m \prod_{i=1}^m \sigma_{i,j}(1)$. By Lemma 2, this equals $\prod_{i\in[m]} \prod_{j=0}^m \sigma_{ij}(1)$. By the definition of $\sigma_{i,0}$ this in turn equals $\prod_{i\in[m]} \pi_i(1)$, as desired.

Perfect Semi-Honest Security: A protocol with the UNIMPC structure is always perfectly semi-honest secure as long as the aggregator is honest, or if all the input parties are corrupt. Hence we

focus on the case when the aggregator P_0 is corrupt and there is at least one honest party. Suppose the adversary corrupts P_0 and $\{P_i \mid i \in S\}$ for some set $S \subsetneq [m]$. Below, we write $\overline{S} := [m] \setminus S$ to denote the set of indices of the honest parties. Recall that an execution of the protocol (including the inputs) is fully determined by the $m \times (m+1)$ matrix $\boldsymbol{\sigma}$, with $(i,j)^{\text{th}}$ entry $\sigma_{ij} \in G_i$, for $(i,j) \in [m] \times ([m] \cup \{0\})$. The input determined by $\boldsymbol{\sigma}$ is defined by $\mathsf{input}(\boldsymbol{\sigma}) = (\pi_1, \cdots, \pi_m)$, where $\pi_i = \prod_{j=0}^{m} \sigma_{ij}$. We say that $\boldsymbol{\sigma}$ is valid if for every $j \in [m]$, $\prod_{i \in [m]} \sigma_{ij}(1) = 1$.

When the functionality is invoked with inputs $\boldsymbol{\pi} = (\pi_1, \cdots, \pi_m)$, in the ideal world, the adversary learns only the corrupt parties' inputs $\boldsymbol{\pi}|_S$ and the residual function of the honest parties' inputs $\pi_{\overline{S}}(1)$, where $\pi_{\overline{S}} := \left(\prod_{i \in \overline{S}} \pi_i\right)$. But in the real world its view consists also $\langle \boldsymbol{\sigma} \rangle_S := \{\sigma_{ij} \mid i \in S \vee j \in S \cup \{0\}\}$. We need to show that for any two input vectors $\boldsymbol{\pi}, \boldsymbol{\pi}'$ with identical ideal views for the adversary – i.e., $\boldsymbol{\pi}|_S = \boldsymbol{\pi}'|_S$, and $\pi_{\overline{S}}(1) = \pi'_{\overline{S}}(1)$ – the distribution of $\langle \boldsymbol{\sigma} \rangle_S$ is also identical. For this we shall show a bijective map $\phi_S^{\boldsymbol{\pi}'}$ between valid matrices $\boldsymbol{\sigma}$ consistent with $\boldsymbol{\pi}$ and those consistent with $\boldsymbol{\pi}'$, which preserves $\langle \boldsymbol{\sigma} \rangle_S$. Since $\boldsymbol{\sigma}$ is distributed uniformly over all valid matrices consistent with the input in the protocol, this will establish that the distribution of $\langle \boldsymbol{\sigma} \rangle_S$ is identical for $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$. More precisely, the following claim completes the proof.

Claim 2. *For any $S \subsetneq [m]$, and any $\boldsymbol{\pi}, \boldsymbol{\pi}' \in G_1 \times \cdots \times G_m$ such that $\boldsymbol{\pi}|_S = \boldsymbol{\pi}'|_S$ and $\boldsymbol{\pi}_S(1) = \boldsymbol{\pi}'_S(1)$, there is is a bijection $\phi_S^{\boldsymbol{\pi}'}$ from $\{\boldsymbol{\sigma} \mid \mathsf{input}(\boldsymbol{\sigma}) = \boldsymbol{\pi} \wedge \boldsymbol{\sigma}$ valid$\}$ to $\{\boldsymbol{\sigma} \mid \mathsf{input}(\boldsymbol{\sigma}) = \boldsymbol{\pi}' \wedge \boldsymbol{\sigma}$ valid$\}$, such that $\langle \boldsymbol{\sigma} \rangle_S = \langle \phi_S^{\boldsymbol{\pi}'}(\boldsymbol{\sigma}) \rangle_S$.*

Proof. Let $S, \boldsymbol{\pi}, \boldsymbol{\pi}'$ be as in the lemma. We shall first define $\phi_S^{\boldsymbol{\pi}'}$ for all $m \times (m+1)$ matrices $\boldsymbol{\sigma}$, with $\sigma_{ij} \in G_i$, and then prove the claimed properties when restricted to the domain in the claim. Fix $h \in \overline{S}$ as (say) the smallest index in $\overline{S}$. Given $\boldsymbol{\sigma}$, $\phi_S^{\boldsymbol{\pi}'}$ maps it to $\boldsymbol{\sigma}'$ as follows.

$$\sigma'_{ij} = \begin{cases} \sigma_{ij} & \text{if } j \neq h \\ \alpha_i^{-1} \circ \pi'_i \circ \beta_i^{-1} & \text{if } j = h \end{cases}$$

where $\alpha_i := \prod_{j=0}^{h-1} \sigma_{ij}$ and $\beta_i := \prod_{j=h+1}^{m} \sigma_{ij}$. Note that like $\boldsymbol{\sigma}$, $\boldsymbol{\sigma}'$ also satisfies the condition that $\sigma'_{ij} \in G_i$ for all $j = 0 \cup [m]$, because $\alpha_i, \beta_i, \pi'_i \in G_i$.

By construction, $\prod_{j=0}^{m} \sigma'_{ij} = \pi'_i$, and hence the image of $\phi_S^{\boldsymbol{\pi}'}$ is contained in $\{\boldsymbol{\sigma}' \mid \mathsf{input}(\boldsymbol{\sigma}') = \boldsymbol{\pi}'\}$. Also, when the domain is $\{\boldsymbol{\sigma} \mid \mathsf{input}(\boldsymbol{\sigma}) = \boldsymbol{\pi}\}$, the mapping is invertible since $\phi_S^{\boldsymbol{\pi}}(\phi_S^{\boldsymbol{\pi}'}(\boldsymbol{\sigma})) = \boldsymbol{\sigma}$, when $\mathsf{input}(\boldsymbol{\sigma}) = \boldsymbol{\pi}$. Hence, by symmetry, this is a bijection from $\{\boldsymbol{\sigma} \mid \mathsf{input}(\boldsymbol{\sigma}) = \boldsymbol{\pi}\}$ to $\{\boldsymbol{\sigma} \mid \mathsf{input}(\boldsymbol{\sigma}) = \boldsymbol{\pi}'\}$. Further, for $i \in S$, $\pi_i = \pi'_i$ and hence $\sigma'_{ih} = \sigma_{ih}$, so that $\langle \boldsymbol{\sigma}' \rangle_S = \langle \boldsymbol{\sigma} \rangle_S$.

It remains to prove that the map is a bijection when the domain and range are restricted to *valid* matrices. So, suppose $\boldsymbol{\sigma}$ is a valid matrix. Then we have

$$\left(\prod_{i \in [m]} \sigma_{ij}\right)(1) = 1 \qquad \forall j \in [m] \tag{7}$$

$$\left(\prod_{i \in [m]} \beta_i\right)(1) = \left(\prod_{j=h+1}^{m} \prod_{i \in [m]} \sigma_{ij}\right)(1) = 1. \tag{8}$$

where the first equality in (8) is obtained by applying Lemma 2, and the second by applying the validity condition (7) successively for $j = m, \cdots, h+1$.

To verify that $\boldsymbol{\sigma}' = \phi_S^{\boldsymbol{\pi}'}(\boldsymbol{\sigma})$ is valid, we only need to verify that $(\prod_{i \in [m]} \sigma'_{ih})(1) = 1$ (as the other columns of $\boldsymbol{\sigma}'$ are the same as in $\boldsymbol{\sigma}$). This we show as follows (where for brevity, we write

$\alpha := \prod_{i\in[m]} \alpha_i$ and $\beta := \prod_{i\in[m]} \beta_i$):

$$\begin{aligned}
&\prod_{i\in[m]} \pi'_i(1) = \prod_{i\in[m]} \pi_i(1) \\
&\quad \Rightarrow (\prod_{i\in[m]} \alpha_i \circ \sigma'_{ih} \circ \beta_i)(1) = (\prod_{i\in[m]} \alpha_i \circ \sigma_{ih} \circ \beta_i)(1) \\
&\quad \Rightarrow \alpha \circ (\prod_{i\in[m]} \sigma'_{ih}) \circ \beta(1) = \alpha \circ (\prod_{i\in[m]} \sigma_{ih}) \circ \beta(1) && \text{by Lemma 2} \\
&\quad \Rightarrow (\prod_{i\in[m]} \sigma'_{ih}) \circ \beta(1) = (\prod_{i\in[m]} \sigma_{ih}) \circ \beta(1) && \alpha \text{ a permutation} \\
&\quad \Rightarrow (\prod_{i\in[m]} \sigma'_{ih})(1) = (\prod_{i\in[m]} \sigma_{ih})(1) = 1 && \text{by (8) and (7).}
\end{aligned}$$

□

Theorem 4. *Any CPS functionality with 4 or fewer parties has a UNIMPC protocol with perfect security. Further, any CPS functionality with 3 or fewer parties has a UNIMPC* protocol with perfect security.*

Proof of Theorem 4. We consider different cases depending on the number of parties. For the first two cases we present UNIMPC* protocols and for the last case a UNIMPC protocol (with an interactive preprocessing phase).

Two parties: Let P_0 be the aggregator and P_1 be the other party. In this case P_1 can simply compute the output and send it to P_0.

Three parties: W.l.o.g., we assume that the given function has no two equivalent inputs for any party. In particular, for $\pi, \pi' \in X_2$ we have $\pi(1) \neq \pi'(1)$.

In this case we can use *any* PSM protocol [FKN94], except that all the randomness is sampled by P_1 and sent to P_2.[6] Clearly, this is a UNIMPC* protocol by construction (and hence is secure when P_0 is honest). When P_1 and P_2 are both honest, the security follows from the original PSM protocol. When P_0 colludes with one of the parties, say, P_1, then note that the adversary in the ideal world learns the output $\pi_1 \circ \pi_2(1)$ as well as π_1, where π_i is the input of party P_i. Since π_1 is a permutation, this determines $\pi_2(1)$, and since we had removed redundant inputs, π_2 itself. Hence in this case a perfect simulation is obtained by a simulator who first finds out P_2's input and then carries out the entire protocol execution.

Four parties: In this case we will rely on an NIMPC protocol $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ for the given CPS functionality (see Appendix A), and any 3-party perfectly secure protocol for general functions that is secure against passive corruption of 1 party (e.g., the passive-secure protocol in [BGW88]). We remark that this transformation has also appeared in a recent, independent work [HIKR18].

1. *Offline phase:* Parties P_1, P_2, P_3 run the general MPC protocol to sample the random variables $(r_1, \cdots, r_m)$ according to Gen. Note that this phase does not need their inputs.
2. *Online phase:* This is identical to the online phase in the NIMPC protocol. Each P_i sends $z_i := \mathrm{Enc}(x_i, r_i)$ to P_0 and P_0 outputs $\mathrm{Dec}(z_1, \cdots, z_m)$.

[6] As a concrete example, to compute an $(n, 2)$-CPS, P_1 can pick a random permutation $\sigma \in S_n$ and send it to P_2. Then P_1 sends the permutation $\pi_1 \circ \sigma^{-1}$ and P_2 sends the value $\sigma \circ \pi_2(1)$ to P_0 who evaluates the former on the latter to obtain the output.

To see that this is secure, we consider the following cases. If only one of P_1, P_2, P_3 is corrupt, then the randomness generation remains secure, and hence the view of the corrupt party (say P_i) in that phase can be computed from r_i. The rest of the view can be simulated by invoking the NIMPC simulation.

If two of P_1, P_2, P_3 are corrupt, then the adversary is allowed to learn the residual function of the remaining party, which is its input (after having removed redundancies). Hence, a perfect simulation is possible in this case as well. □

7 Latin Hypercubes

CPS functions are closely related to Latin Squares, and more generally, *Latin Hypercubes*. An n-ary Latin Square is an $n \times n$ matrix with entries from $[n]$ such that each row and column has all elements of $[n]$ appearing in it. The m-dimensional version is similarly a tensor indexed by m-dimensional vectors, so that every "row" (obtained by going through all values for one coordinate of the index, keeping the others fixed) is a permutation of $[n]$.

Formally, we may define Latin squares and hypercubes in terms of "codes." Here, one represents a Latin square M using the set $L_M = \{(i, j, M_{ij}) \mid i, j \in [n]\}$. Restricted to any two coordinates, the n^2 entries in L form the set $[n] \times [n]$; or equivalently, there are no tuples in this set which differ in exactly one coordinate. The m-dimensional version can be defined as follows (where Δ_{H} stands for Hamming distance):

Definition 11. *$L \subseteq [n]^{m+1}$ is said to be an m-dimensional, n-ary* Latin hypercube *if $|L| = n^m$ and for any $x, x' \in L$, $\Delta_{\mathrm{H}}(x, x') \neq 1$.*

An $(m+1)$-party aggregating functionality $f : [n]^m \to [n]$ is said to be a Latin hypercube functionality *if $\{(x_1, \cdots, x_m, f(x_1, \cdots, x_m)) \mid (x_1, \cdots, x_m) \in [n]^m\}$ is an n-ary Latin hypercube.*

In the case of $m = 2$, an n-ary Latin square functionality f always is (or, technically, embeds into) an $(n, 2)$-CPS (X_1, X_2).[7] However, this is not true in higher dimensions (see Appendix C for an explicit example of a function that is a Latin hypercube, but not a CPS). So not all Latin hypercube functions can have MPC protocols. We obtain an exact characterization of all Latin hypercube functionalities that have UNIMPC* (or MPC) protocols.

Theorem 5. *A Latin hypercube functionality has a UNIMPC* protocol if and only if it is a CPS functionality.*

Recall that by Theorem 2 only CPS functionalities can have even MPC protocols. Hence the above theorem also characterizes the Latin hypercube functionalities that have an MPC protocol.

To prove the "if" direction of the above theorem, we shall relate Latin hypercubes which are CPS functionalities, to *Complete* CPS functionalities, and then further discover an additional "full commutativity" structure in such functions. This additional structure is used to show that such functions are CPSS functionalities, letting us use the UNIMPC* protocol from Theorem 3.

[7] We let $X_1 = \{\pi_i \mid \pi_i(f(1,j)) = f(i,j)\ \forall j \in [n]\}$, and $X_2 = \{\rho_j \mid \rho_j(f(i,1)) = f(i,j)\ \forall i \in [n]\}$. These functions are well-defined permutations because of f being a Latin square functionality, and it is a CPS because, $\pi_i \circ \rho_j(f(1,1)) = \rho_j \circ \pi_i(f(1,1)) = f(i,j)$. With a bijective embedding that relabels the outputs of f so that $f(1,1) = 1$, this meets the definition of a CPS.

7.1 Latin Property and Completeness

A Latin Hypercube can alternately be defined as a function satisfying two properties – *Latin Property* and *Completeness*. The Latin property for a function $f : X_1 \times \cdots \times X_m \to [n]$ requires that there should not exist two inputs $(x_1, \cdots, x_m)$ and $(x'_1, \cdots, x'_m)$ which have a Hamming distance of 1 and $f(x_1, \cdots, x_m) = f(x'_1, \cdots, x'_m)$. We point out that the Latin property is a necessary (but not sufficient) condition for a function to be a CPS (after removing redundant inputs): If $f(\pi_1, \cdots, \pi_m) = f(\pi'_1, \cdots, \pi'_m)$ and for all $j \neq i$, we have $\pi_j = \pi'_j$, then we have $\pi_i(1) = \pi'_i(1)$, which in turn means that π_i and π'_i are equivalent inputs. Also recall that CPS captures the condition that any set of parties learning the output (along with their own inputs) should be able to learn the residual function of the remaining parties (Theorem 2); Latin property is equivalent to the (weaker) condition that if a collusion containing all but one input player learns the output, then they can learn the remaining player's input.

The second property of completeness was originally defined for CPS functionalities in Definition 9, but we now extend it to functions with the Latin property: Note that if $f : X_1 \times \cdots \times X_m \to [n]$ has the Latin property, then $|X_i| \leq n$ for all i; we say that f is complete if $|X_i| = n$ for all $i \in [m]$.

Then, a Latin Hypercube function is simply a function which has both the Latin property and the completeness property. Since a CPS always has the Latin property, a Complete CPS is the same as a Latin hybercube that is a CPS.

7.2 Fully Commuting Permutations System

Our goal from above is to show that a Complete CPS functionality has the subgroup structure of a CPSS functionality. In this section we do this by showing a larger class of functionalities which embed into CPSS functionalities.

Definition 12. *An* (n, m)-Fully Commuting Permutations System *(FCPS) is a collection* $(X_1, \cdots, X_m)$ *where for all* $i \in [m]$, $X_i \subseteq S_n$ *is non-empty, and for any two distinct* $i, j \in [m]$ *and* $\pi \in X_i$ *and* $\pi' \in X_j$, $\pi \circ \pi' = \pi' \circ \pi$.

It is called an (n, m)-Fully Commuting Permutation Subgroups System *(FCPSS) if each* X_i *is a subgroup of* S_n.

Lemma 4. *For every* (n, m)*-FCPS* $(X_1, \cdots, X_m)$ *there is an* (n, m)*-FCPSS* $(G_1, \cdots, G_m)$ *such that for* $i \in [m]$, $X_i \subseteq G_i$.

Proof. We define G_i to be the subgroup of S_n generated by X_i. Note that each G_i is finite (since S_n is finite) and is obtained from X_i by iteratively adding to it π^{-1} for some $\pi \in X_i$, or $\pi_1 \circ \pi_2$ for $\pi_1, \pi_2 \in X_i$. So it is enough to prove that one step in this iterative process preserves the commuting property. If $\pi \in X_i$, then for any $j \neq i$ and $\pi' \in X_j$, we have

$$\pi \circ \pi' = \pi' \circ \pi \Rightarrow \pi^{-1} \circ (\pi \circ \pi') \circ \pi^{-1} = \pi^{-1} \circ (\pi' \circ \pi) \circ \pi^{-1} \Rightarrow \pi' \circ \pi^{-1} = \pi^{-1} \circ \pi'.$$

Also, for $\pi_1, \pi_2 \in X_i$ and $\pi' \in X_j$, for $i \neq j$, we have

$$(\pi_1 \circ \pi_2) \circ \pi' = \pi_1 \circ (\pi' \circ \pi_2) = \pi' \circ (\pi_1 \circ \pi_2).$$

Thus adding π^{-1} and $\pi_1 \circ \pi_2$ to X_i retains the commuting property. □

The following lemma relates completeness with full-commutativity.

Lemma 5. *If a CPS is complete in at least 3 dimensions, then it is an FCPS.*

Proof. Suppose $(X_1, \cdots, X_m)$ is an (n, m)-CPS which is complete in three dimension. Consider any two distinct $i, j \in [m]$, and $\pi \in X_i$, $\pi' \in X_j$. We need to show that $\pi \circ \pi' = \pi' \circ \pi$. Let $k \in [m] \setminus \{i, j\}$ be a dimension in which the CPS is complete. Then, by the CPS property, for each $\rho \in X_k$, $\pi \circ \pi' \circ \rho(1) = \pi' \circ \pi \circ \rho(1)$. Since $\{\rho(1) \mid \rho \in X_k\} = [n]$ we have that for every $a \in [n]$, $\pi \circ \pi'(a) = \pi' \circ \pi(a)$, or in other words, $\pi \circ \pi' = \pi' \circ \pi$, as required. □

Interestingly, in the above lemma the number of dimensions 3 is tight, as demonstrated by Latin squares (which are complete in 2 dimensions). In Appendix C we give an example of a Latin square (i.e., an $(n, 2)$-CCPS functionality) that is not an FCPS (or even in **CPSS**).

The above results on FCPS lead us to our result that Latin hypercubes which are CPS (i.e., CCPS) are also CPSS.

Lemma 6. *For $m > 2$, an (n, m)-CCPS is an (n, m)-CPSS.*

Proof. By Lemma 5, for $m > 2$, an (n, m)-CCPS is an FCPS, and hence embeddable in an FCPSS (by Lemma 4). However, since the given function is already complete in all dimensions, being embeddable in an FCPSS translates to being an FCPSS itself. Thus the given CCPS is also an FCPSS, and in particular, a CPSS. □

Finally, we can prove Theorem 5.

Proof of Theorem 5. The "only if" direction follows from Theorem 2. We need to argue that every Latin hypercube functionality that is a CPS functionality has a UNIMPC* protocol. If the functionality has up to 3 parties, it follows from Theorem 4 that it has a UNIMPC* protocol. For functionalities with 4 or more parties, we note that the functionality corresponds to a Latin hypercube of 3 or more dimensions, i.e., an (n, m)-CCPS for $m \geq 3$. Then by Lemma 6 it is a CPSS functionality and hence by Theorem 3, the functionality has a UNIMPC* protocol. □

8 Towards a Characterization of Strong Security

While security against active corruption is often stronger than security against passive corruption, this is not always the case. This is because, in the ideal world model for active corruption, the adversary (i.e., simulator) is allowed to send any inputs of its choice to the functionality, the adversary in the passive corruption setting is required to send the same input as the corrupt parties received. To reconcile this discrepancy, one could weaken the notion of passive security by allowing the simulator to change the input sent to the functionality. However, the resulting security guarantee is quite pessimistic, as it assumes that even passively corrupt parties will alter their inputs, and may not be appropriate in scenarios where the passively corrupt parties will not do so (see Footnote 3). Instead, we propose using a stronger definition – which we simply call *strong security* – which requires the simulator to not alter the inputs if the parties are corrupted passively, but allows it to use arbitrary inputs if they are corrupted actively. Formally, we use the following information-theoretic security definition:

Definition 13 (Strong security)**.** *A protocol Π is said to be a* strongly secure *protocol for a functionality $\mathcal{F}$ if it is both passive secure and UC secure (with selective abort) for $\mathcal{F}$ against computationally unbounded adversaries.*

Note that strong security admits composition as both semi-honest security and UC security are composable. From a practical point of view, strong security (possibly weakened to hold only against PPT adversaries) is important, and arguably the "right" notion in many cases. Here we initiate the study of characterizing multi-party functionalities that are strongly securely realizable. Clearly, the impossibility results for both UC security and passive security apply to strong security.

To state our results for *all* multi-party functions, we need to go beyond aggregating functionalities. Firstly, we shall need the notion of disseminating functionalities: An $(m+1)$-party disseminating functionality $f = (f_1, \cdots, f_m)$ has a single party P_0 with an input x, so that every other party P_i receives the output $f_i(x)$. The class of disseminating functions is denoted by **DISS**. Secondly, we need to consider functions which are "essentially" aggregating or disseminating, but not strictly so because of the presence of additional information in each party's local output which is derived solely from its own inputs. The idea that a function can be *essentially the same* as another function is captured using the notion of isomorphism among functionalities, as defined in [MPR13]. We reproduce this below, adapted to strong security. Here, a protocol $\pi^{\mathcal{G}}_{\mathcal{F}}$ for $\mathcal{F}$, using $\mathcal{G}$ as a setup, is said to be *local* if each party (deterministically) maps its input to an input for the functionality $\mathcal{G}$, then calls $\mathcal{G}$ once with that input and, based on their private input and the output obtained from $\mathcal{G}$, locally computes the final output (deterministically), without any other communication.

Definition 14 (Isomorphism [MPR13])**.** *We say $\mathcal{F}$ and $\mathcal{G}$ are* isomorphic *to each other if there exist two local protocols $\pi^{\mathcal{G}}_{\mathcal{F}}$ and $\pi^{\mathcal{F}}_{\mathcal{G}}$ that strongly securely realize $\mathcal{F}$ and $\mathcal{G}$ respectively.*

Now we are ready to state and prove our main results regarding strongly secure MPC.

Theorem 6. *If a functionality has a strongly secure protocol, then it is isomorphic to a functionality in* **DISS** $\cup$ **CPS**.

Proof. It follows from [PR08a] that all strongly securely realizable functionalities are isomorphic to a *disseminating* functionality (i.e., a functionality in **DISS**), or an *aggregating* functionality (as defined in here).Further, if a functionality $\mathcal{F}$ that has a strongly secure protocol is isomorphic to an aggregating functionality $\mathcal{F}'$, then from the definition of isomorphism, $\mathcal{F}'$ too has a strongly secure (and in particular, a passive secure) protocol. Then, by Theorem 2, $\mathcal{F}' \in$ **CPS**. □

We contrast this with our positive result below, which refers to **CCPS** (Definition 9), instead of **CPS**. We point out that our protocols below are efficient in the sense of having polynomial complexity in the statistical security parameter, but can be polynomial (rather than logarithmic) in the domain sizes or exponential in the number of parties.

Theorem 7. *If a functionality is isomorphic to one in* **DISS** $\cup$ **CCPS**, *then it has a strongly secure protocol.*

Proof. We show in Section 8.2 that every disseminating functionality has a UC secure protocol. A UC secure protocol for a disseminating functionality is always passive secure as well: only the disseminator has any input, and if the disseminator is passively corrupt, the correctness guarantee under UC security (when no party is corrupt) ensures that the simulator can send the disseminator's actual input to the functionality.

In Lemma 7, we prove that the UNIMPC* protocol in Section 6 is UC secure for every Complete CPSS functionality. By Lemma 6, this covers all Complete CPS functionalities of more than 2 dimensions. For 2-dimensional Complete CPS functionalities (which are precisely Latin Squares),

we give a UC secure protocol in Appendix B. In Section 8.1, we show a compiler that extends these results to functionalities embedded in a CCPS functionality.

Finally, we note that for aggregating CPS functionalities too, UC security implies strong security: If the aggregator is honest, the correctness guarantee under UC security allows the simulator to send the corrupt parties' actual input to the functionality; if the aggregator is corrupt, a simulator which sends the correct inputs of the passively corrupt players obtains the honest parties' residual function, and can internally execute the UC simulator (which may send arbitrary inputs to the functionality and expect the output). □

To complete the proof above, we need to prove the following lemma.

Lemma 7. *The UNIMPC* protocol in Section 6 is UC secure for any complete CPSS functionality.*

Proof. If the aggregator is corrupt its behavior can be mimicked in the ideal world. So, we describe the case where the aggregator is honest.

Since we have already shown that the protocol is secure against a semi-honest adversary, we will simply show that any execution of an actively malicious adversary corresponds to an execution of an equivalent semi-honest adversary such that the view of the honest parties in both the cases remains the same.

More formally we define a hybrid world $\mathcal{H}$ where the adversary is semi-honest corrupt. In the real world $\mathcal{R}$ the adversary is actively corrupt. Let T be the set of corrupt parties. We can show that $\forall \mathcal{A}_R$, $\exists \mathcal{A}_H : \text{VIEW}_{x_{\overline{T}}}(\{P_i | i \in \overline{T}\})$ is the same for both the worlds.

In the randomness computation phase, the malicious adversary may choose a $U \subset T$ and $\sigma_{1j}, \ldots, \sigma_{mj}, \forall j \in U$, such that $\sigma_{1j} \circ \cdots \circ \sigma mj(1) = \lambda_j$ where $\lambda_j \neq 1$. We describe an equivalent semi-honest adversary which chooses

$$\sigma'_{ij} = \begin{cases} \sigma_{ij} & \text{if } i \neq j \\ \pi_i \in G_i : \pi_i(\lambda_i) = 1 \text{ where } \lambda_i = \sigma_{1i} \circ \cdots \circ \sigma_{i-1,i} \circ \sigma_{i+1,i} \circ \cdots \circ \sigma_{mi}(1) & \text{otherwise} \end{cases}$$

By the completeness property we know that $\exists \pi'_i \in G_i : \pi'_i(1) = \lambda_i$. We simply define $\pi_i = \pi_i'^{-1}$. In both $\mathcal{H}$ and $\mathcal{R}$, $\text{VIEW}_{x_{\overline{T}}}(\{P_i | i \in \overline{T}\}) = \sigma_{i,j}, i, j \in \overline{T}$ which is the same in both the cases. We can see that the set $\sigma'_{ij} \forall j$ acts as a stabilizer of 1.

In the input generation phase the adversary can send some $\tau_i \in G_i$ to the aggregator instead of σ_{i0}. For a semi honest adversary this corresponds to $\pi'_i = \tau_i \circ \sigma_{i0}^{-1} \circ \pi_i$ which is valid due to the closure property. □

8.1 Restricting Input Domains While Retaining UC Security

In this section we give a compiler to transform a UC secure protocol for a CPS functionality $\mathcal{F}$ to a UC secure protocol for the same functionality, but with restricted input domains for each party. To illustrate the need for this compiler, suppose m input parties wish to total their votes (0 or 1) and provide it to an aggregator, securely. We do have a UC secure protocol for addition modulo $m+1$, and this functionality can correctly compute the total of m bits. However, this is not a UC secure protocol for our functionality, as the corrupt parties can provide inputs other than 0 or 1. Nevertheless, we show that the original protocol can be transformed into one which restricts the domain as desired.

Definition 15 (Domain Restriction)**.** *Given a functionality $\mathcal{F}$ with input domain $X = X_1 \times \cdots \times X_m$, we define a* domain restriction of $\mathcal{F}$ to $D = D_1 \times \cdots \times D_m \subseteq X$ *as a functionality $\mathcal{F}_D$ which is defined only on inputs in D, where it behaves identically as $\mathcal{F}$.*

We give a compiler that transforms a UC secure protocol for a CPS functionality $\mathcal{F}$ to a UC secure protocol for $\mathcal{F}_D$ for any $D = D_1 \times \cdots \times D_m$. Our compiler can be presented as a protocol $\mathsf{RDom}_D^{\mathcal{F},\mathcal{F}_{\text{AND}}}$ – a protocol in a hybrid model with access to the ideal functionalities $\mathcal{F}$ and (m-input) aggregating functionality $\mathcal{F}_{\text{AND}}$. We note that while $\mathcal{F}_{\text{AND}}$ is not a CPS functionality (and hence cannot have a passive secure protocol), it does have a UC secure protocol. Specifically, one can reduce $\mathcal{F}_{\text{AND}}$ to summation over an exponentially large abelian group, where each party P_i maps its input x_i to a group element g_i as follows: if $x_i = 0$, let g_i be random, and if $x_i = 1$, let $g_i = 0$. The aggregator receives $\sum_i g_i$ and outputs 1 if the sum is 0, and 0 otherwise.

Protocol $\mathsf{RDom}_D^{\mathcal{F},\mathcal{F}_{\text{AND}}}$

The high-level idea of this protocol is to first invoke $\mathcal{F}$ on random inputs from the domain D, and use a cut-and-choose phase to verify that indeed most of the invocations used inputs in the domain D. Then, using access to $\mathcal{F}_{\text{AND}}$, the executions involving the correct input from all the parties are isolated, and the aggregator P_0 outputs what it received from $\mathcal{F}$ in those executions (if there is a consistent output). The formal description follows.

Let $\mathcal{F}$ represent the functionality to be realized and k be the security parameter. Let $\mathcal{E}$ be the input domain of $\mathcal{F}$ and D be the desired domain. Let $P_i, i \in [m] \cup \{0\}$ be the set of parties with inputs $\{x_i\}_{i\in[m]}$. Let P_0 be the aggregator with output space $[n]$.

1. **Random Execution:** Invoke k sessions of the functionality $\mathcal{F}$ with domain $\mathcal{E}$. Each honest party $P_i, i \in [m]$ chooses input uniformly at random from domain D. Let u_{ij} be the input used by party P_i in the j^{th} execution and let v_j be its output.
2. **Opening:** P_0 chooses $S \subseteq [k]$, where every element has a probability of 0.5 of being picked up (thus $\mathbb{E}(|S|) = k/2$), and announces it. Every party $P_i, i \in [m]$ sends $u_{ij}, \forall j \in S$ to P_0. Then, P_0 checks the consistency of all the inputs and outputs it received: i.e., if $\forall j \in S, \mathcal{F}(\{u_{ij}\}_{i\in[m]}) = v_j$. It also confirms that each input is chosen from the domain D. Otherwise P_0 aborts.
3. **Tallying with actual inputs:** Invoke $k - |S|$ sessions of the $\mathcal{F}_{\text{AND}}$ functionality, indexed by $\bar{S} = [m] \setminus S$. Each honest party P_i sets its input to session j of $\mathcal{F}_{\text{AND}}$ a_{ij} as
$$a_{ij} = \begin{cases} 1 & \text{if } v_{ij} = x_i \\ 0 & \text{otherwise} \end{cases}$$
and let the output for j^{th} $\mathcal{F}_{\text{AND}}$ be b_j. Also let $T = \{j : b_j = 1\}$.
4. **Computing the result:** If $|T| \geq t/2$ where $t = k/(2 \cdot \prod_{i\in[m]} |X_i|)$ is the expected size of T, and if $\exists v \forall j \in T$, $v_j = v$, then P_0 outputs v. Otherwise P_0 Aborts.

Theorem 8. *If $\mathcal{F}$ is an m-input CPS functionality, and $D = D_1 \times \cdots \times D_m$ is a subset of its domain, then $\mathsf{RDom}_D^{\mathcal{F},\mathcal{F}_{\text{AND}}}$ is a UC secure protocol for $\mathcal{F}_D$.*

Proof. We need to give simulators for various subsets of corruption. We start by considering the case when all the parties are honest. In this case, we need only show that the protocol produces correct outputs on all inputs, except with negligible probability. Note that if all the parties are honest, P_0

can abort only if $|T| < t/2$ in Step 4, where t is the expected number of instances that fall into the set T. Since each instance falls into T independently, we can bound this probability using Chernoff bound, to be exponentially small in t (which is in turn, linear in the security parameter). This is summarized below.

Lemma 8. *If all parties P_i are honest, then the probability of abort when the input is picked by the parties accordingly from their respective domains is a negligible function of the security parameter.*

If the aggregator is corrupt a simulator can send any input to the functionaliy on behalf of the corrupt parties, and obtain the residual function of the honest parties. Then it picks an arbitrary valid input combination for the honest parties which has that residual function, and runs the honest parties' protocol with that input. It can be seen that this is a perfect simulation.

The interesting case is when the aggregator is honest. We describe a simulator for this setting. The simulator first carries out the random execution phase and the opening phase faithfully, on behalf of the honest players. If these phases do not result in an abort, it proceeds as follows, depending on the number of random execution instances c in which the input combination $x_{\overline{H}}$ sent by the corrupt parties to $\mathcal{F}$ do not have an equivalent input in the restricted domain D. (Below, $t = \Omega(k)$ is as specified in the protocol.)

Case 1 : $c \geq t/2$. This case will occur with negligible probability. Note that if one of the instances in which there is no equivalent valid input exists is chosen for opening, then the aggregator will abort, no matter how the corrupt parties explain their inputs (because, the function being a CPS, any consistent explanation by the corrupt parties should have the same residual function as their original input, and no such explanation exists in the restricted domain). The probability that none of the c invalid instances were chosen is $\frac{1}{2^c}$, which is negligible in this case.

Case 2 : $c < t/2$. In this case the simulator carries out a simulation of the tallying phase using an arbitrary input in the restricted domain for the input players. If this simulated execution aborts, the simulator instructs $\mathcal{F}_D$ to abort. Otherwise, $|T| \geq t/2$ and there must exist a value v such that for all $j \in T$ in the simulated execution, $v_j = v$. Since $c < t/2$, at least one of these instances corresponds to when the corrupt parties inputs has an equivalent input $x^*_{\overline{H}} \in D$; in fact, then, the function being a CPS, in all such instances, the corrupt parties' inputs are equivalent to each other. The simulator sends $x^*_{\overline{H}}$ to $\mathcal{F}_D$, completing the simulation. □

8.2 Disseminating Functionalities

We rely on the disseminated-OR functionality $\mathcal{D}_{\text{OR}}$ to show that all disseminated functionalities are UC secure. The functionality $\mathcal{D}_{\text{OR}}$ takes $(x_1, \cdots, x_m)$ from the disseminator P_0 and outputs (b, x_i) to P_i where $b = x_1 \vee \cdots \vee x_m$. We start by giving a UC secure protocol for $\mathcal{D}_{\text{OR}}$.

Protocol for $\mathcal{D}_{\text{OR}}$. In [PR08a] a UC secure protocol for 3-party $\mathcal{D}_{\text{OR}}$ was given. We present a variant that works for all values of m.

1. P_0 broadcasts (UC-securely [GL02]) $b := \bigvee_{i>0} x_i$ to all P_i.
2. If $b = 0$, for each $i > 0$, P_i outputs $(0, 0)$ and halts. Else, they continue.
3. P_0 sends x_i to each P_i.
4. For $i \in [m]$, $\ell \in [k]$, P_0 samples $r_{i\ell}$ from a large group (e.g., k-bit strings) s.t. $\forall \ell$, $\sum_i r_{i\ell} = 0$.
5. For each i, if $x_i = 0$, P_0 sends $r_{i\ell}$ for all ℓ to P_i (and otherwise sends nothing to P_i).
6. Cut-and-choose:

(a) P_1 picks a random subset $S \subset [k]$ of size $k/2$ and sends it to P_0.
(b) For all $\ell \in S$, P_0 broadcasts $r_{i\ell}$ for all i, and all parties verify that $\sum_i r_{i\ell} = 0$. P_1 verifies that the set S used is what it picked.
(c) Any P_i with $x_i = 0$ aborts if it sees that for some ℓ, $r_{i\ell}$ broadcast by P_0 is not equal to $r_{i\ell}$ it received.

7. For each $\ell \notin S$, $P_1, \cdots, P_m$ do the following:
   (a) For each i, if $x_i = 0$, P_i sets $s_{i\ell} = r_{i\ell}$, and otherwise samples $s_{i\ell}$ randomly.
   (b) They use the standard semi-honest secure protocol to compute $\sum_i s_{i\ell}$.
   (c) Each P_i aborts if it gets the sum as 0.
8. If no abort has been observed, each P_i outputs $(1, x_i)$, where x_i is as received from P_0 in the beginning. Otherwise it aborts.

We sketch the proof of UC security of this protocol. Firstly, note that if $b = 0$ in the first step, irrespective of whether P_0 is honest or not, the simulation is easy. So we focus on the case that $b = 1$ in the first step.

If P_0 and one or more of P_i, $i \in [m]$, collude, the simulator can make the honest parties P_j output any value of the form $(1, x_j)$, by simply sending a vector $(x_1, \cdots, x_m)$ to the functionality, with $x_i = 1$ for at least one corrupt party P_i. The remaining cases are as follows:

Only P_0 is corrupt: The simulator extracts P_0's inputs from all the messages x_i it sends to the honest parties P_i in Step 3. If the bit broadcast $b = 1$ and if $\bigvee_i x_i = 1$, the simulator can simply forward these x_i to the functionality. (By perfectly simulating the execution of the honest parties, the simulator can further decide which parties to selectively abort; the ones which do not abort will indeed output x_i as sent to them in Step 3.)

If $b = 1$ and $\bigvee_i x_i = 0$, then the simulator simulates an abort by all the honest parties. To see that this is a good simulation, we consider two cases: There are several (say $> k/2$) "bad" $\ell \in [k]$ such that $\sum_i r_{i\ell} \neq 0$, or not. In the first case, at least one bad ℓ will be chosen by P_1 for the cut-and-choose step, causing all honest parties to abort with all but negligible probability; in the second case, there will be at least one good ℓ left out of the cut-and-choose step, and then, the result of the semi-honest secure protocol in Step 7 (where all P_i use $s_{ij} = r_{ij}$, since they all have $x_i = 0$) will be 0, again causing all parties to abort.

P_0 is honest: Note that if all P_i are corrupt, or if all P_i are honest, simulation is trivial. So suppose a subset of P_i are corrupt. Steps up to 6 can be perfectly simulated knowing only x_i for the corrupt P_i. In Step 7, the use of a semi-honest secure protocol for summation of $s_{j\ell}$ allows that the adversary to learn all $s_{j\ell}$ used by the honest parties. (It can also arbitrarily influence the output of this summation protocol, but this can be perfectly simulated by the simulator once it knows all $s_{j\ell}$, by using selective aborts in the ideal world.)

We claim that all $s_{j\ell}$ can be perfectly simulated by picking them to be random elements. To see that this is secure, there are two cases to consider: All corrupt P_i have $x_i = 0$ (and hence some honest party P_{j^*} has $x_{j^*} = 1$), or some corrupt P_{i^*} has $x_{i^*} = 1$. In the first case, this is because the honest party P_{j^*} sets $s_{j^*\ell}$ to be random. In the second case, the corrupt party P_{i^*} did not receive $r_{i^*\ell}$, rendering all $r_{j^*\ell}$ to be uniformly random. and hence all $s_{j^*\ell}$ also to be uniformly random.

Protocol for any disseminating functionality. A disseminating functionality $\mathcal{F}$ with m output parties is specified by a function $F : X \to Y_1 \times \cdots \times Y_m$, for some finite domains X and

Y_i. We consider a boolean function $\mathrm{Inv}^F_{[m]} : Y_1 \times \cdots \times Y_m \to \{0,1\}$ (for "invalid") as follows: $\mathrm{Inv}^F_{[m]}(y_1, \cdots, y_n) = 1$ iff $\nexists x \in X$ s.t. $F(x) = (y_1, \ldots, y_n)$.

More generally, for any $S \subseteq [m]$, define $\mathrm{Inv}^F_S : Y_S \to \{0,1\}$ as follows (denoting by Y_S the input combinations of parties indexed by S): for $y_S \in Y_S$, $\mathrm{Inv}^F_S(y) = 1$ iff $\nexists x \in X, y_{\overline{S}} \in Y_{\overline{S}}$ s.t. $F(x) = (y_S, y_{\overline{S}})$ (with the output tuple understood as being sorted appropriately by the indices).

Protocol $\mathsf{Diss}^{\mathcal{D}_{\mathrm{OR}}}_{\mathcal{F}}$ (for disseminating functionality $\mathcal{F}$ computing F):

1. On input x, P_0 sends y_i to each P_i, where $F(x) = (y_1, \cdots, y_m)$.
2. For each subset $S \subseteq [m]$
   - For each $\tilde{y}_S \in Y_S$ such that $\mathrm{Inv}^F_S(\tilde{y}_S) = 1$:
     (a) Invoke $\mathcal{D}_{\mathrm{OR}}$, with P_0's input being $(a_1, \cdots, a_m)$, where $a_i = 0$ iff $\tilde{y}_i = y_i$ and 1 otherwise.
     (b) Each P_i receives (b, a_i). If $b = 0$, or if $a_i = 1$ but $\tilde{y}_i = y_i$, then abort.
3. If no abort has been observed, each P_i outputs y_i, and else aborts.

We point out that it is important to have the protocol consider all subsets $S \subseteq [m]$ (which makes it take time exponential in m), and not just the whole set $[m]$, as otherwise P_0 can collude with a corrupt P_{i^*} (who never aborts), and ensure that $b = 1$ always, by setting $a_{i^*} = 1$. Then P_0 can make the honest parties accept any combination of outputs, valid or not.

Below we prove that the above protocol UC securely realizes $\mathcal{F}$.

Lemma 9. *Any disseminated functionality $\mathcal{F}$ is UC securely realized by $\mathsf{Diss}^{\mathcal{D}_{\mathrm{OR}}}_{\mathcal{F}}$.*

Proof. If P_0 is honest, then the following simulation is easily seen to be a perfect simulation: the simulator obtains the output for the corrupt parties $y_{\overline{H}}$, finds an arbitrary input x such that $F(x)_{\overline{H}} = y_{\overline{H}}$, and faithfully executes the protocol with the corrupt players, itself playing the role of P_0 with input x as well as of the other honest parties P_H. It observes which honest parties abort, and let the functionality deliver the output to the others.

The more interesting case is when P_0 is corrupt, and possibly colluding with a set $P_{\overline{H}}$ of output parties. In this case, the simulator obtains y_H from the corrupt P_0. We consider two cases:

- If there exists an x such that $F(x)_H = y_H$: Then, the simulator sends one such x to $\mathcal{F}$, executes the honest parties' protocol faithfullym and observes which ones abort; it lets the functionality deliver the output to the others.
- If no such x exists: Then, the simulator makes all the honest parties abort.

In the first case, the simulation is perfect, since, in the real execution, if any honest party P_i does not abort, it will output the value y_i received in the first round. In the second case, we claim that all the honest parties would abort in the protocol. This is because, $\mathrm{Inv}^F_H(y_H) = 1$, and hence, when $S = H$ in the loop for each subset $S \subseteq [m]$, and $\tilde{y}_S = y_H$, the honest parties will abort (either because all $a_i = 0$ and hence $b = 0$, or because some $a_i = 1$). Thus in this case too, the simulation is perfect. □

Acknowledgments

The last author acknowledges helpful discussions with Frederick Douglas during an undergraduate summer internship in 2012, when a version of the protocol in Appendix A was observed. Prior to that, a version of the protocol $\mathsf{Diss}^{\mathcal{D}_{\mathrm{OR}}}_{\mathcal{F}}$ in Section 8.2 was observed during discussions with Mike Rosulek.

References

[BGI+14] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In *Advances in Cryptology - CRYPTO 2014, Proceedings, Part II*, pages 387–404, 2014.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th STOC*, pages 1–10, 1988.

[Blu81] Manuel Blum. Three applications of the oblivious transfer: Part I: Coin flipping by telephone; part II: How to exchange secrets; part III: How to send certified electronic mail. Technical report, University of California, Berkeley, 1981.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Extended abstract in FOCS 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th STOC*, pages 11–19, 1988.

[CI96] Benny Chor and Yuval Ishai. On privacy and partition arguments. In *Fourth Israel Symposium on Theory of Computing and Systems, ISTCS 1996, Jerusalem, Israel, June 10-12, 1996, Proceedings*, pages 191–194, 1996. Journal version appears in Information and Computation, 167(1).

[CK91] Benny Chor and Eyal Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *DISC*, pages 17–32, 2002.

[HIJ+16] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 157–168, 2016.

[HIJ+17] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In *ASIACRYPT 2017, Proceedings, Part III*, pages 181–211, 2017.

[HIKR18] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Best possible information-theoretic mpc. In *To appear in the Proceedings of Theory of Cryptography - 16th Theory of Cryptography Conference, TCC*, 2018.

[HM97] Martin Hirt and Ueli M. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *PODC*, pages 25–34, 1997.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Israel Symp. Theory of Comp. and Systems, ISTCS*, pages 174–184, 1997.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

[KMR09] Robin Künzler, Jörn Müller-Quade, and Dominik Raub. Secure computability of functions in the IT setting with dishonest majority and applications to long-term security. In *TCC*, pages 238–255, 2009.

[Kus89] Eyal Kushilevitz. Privacy and communication complexity. In *FOCS*, pages 416–421, 1989.

[MPR09] Hemanta Maji, Manoj Prabhakaran, and Mike Rosulek. Complexity of multi-party computation problems: The case of 2-party symmetric secure function evaluation. In *TCC*, pages 256–273, 2009.

[MPR13] Hemanta Maji, Manoj Prabhakaran, and Mike Rosulek. *Complexity of Multi-Party Computation Functionalities*, volume 10 of *Cryptology and Information Security Series*, pages 249 – 283. IOS Press, Amsterdam, 2013.

[OY16] Satoshi Obana and Maki Yoshida. An efficient construction of non-interactive secure multiparty computation. In *Cryptology and Network Security, CANS*, pages 604–614, 2016.

[PP12] Manoj Prabhakaran and Vinod Prabhakaran. On secure multiparty sampling for more than two parties. In *Proceedings of the 2012 IEEE International Information Theory Workshop (ITW 2012)*, 2012.

[PR08a] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In *CRYPTO*, pages 262–279, 2008. Full version available as ECCC Report TR08-050 from https://eccc.weizmann.ac.il.

[PR08b] Manoj Prabhakaran and Mike Rosulek. Towards robust computation on encrypted data. In *ASIACRYPT*, pages 216–233, 2008.

[Rys51] H. J. Ryser. A combinatorial theorem with an application to latin rectangles. *Proceedings of the American Mathematical Society*, 2(4):550–552, August 1951.

[SRA79] Adi Shamir, R. L. Rivest, and Leonard M. Adleman. Mental poker. Technical Report LCS/TR-125, Massachusetts Institute of Technology, April 1979.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computation. In *Proc. 23rd FOCS*, pages 160–164, 1982.

A A General NIMPC Protocol

We give a simple construction of a perfectly secure NIMPC protocol for any function in the information theoretic setting, which is a generalization (and arguably, a more direct presentation) of a protocol in [HIJ+16].

W.l.o.g., let $f : X \to \mathbb{Z}_n$ be the functionality to be realized, where $X = \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_m}$. (We use $\mathbb{Z}_{n_i}$ as input spaces to conveniently define a set of cyclic permutations over them, and $\mathbb{Z}_n$ as the output space to conveniently describe secret-sharing over it.)

1. **Randomness Generation:** $\mathrm{Gen}(f) \to (\rho_1, \cdots, \rho_m)$ as follows. Let $\tilde{f}, \tilde{f}_1, \cdots, \tilde{f}_m : X \to \mathbb{Z}_n$ be s.t., $\forall x = (x_1, \cdots, x_m) \in X$:

$$\tilde{f}(x_1 + \pi_1, \cdots, x_m + \pi_m) = f(x_1, \cdots, x_m)$$
$$\tilde{f}(x) = \tilde{f}_1(x) + \cdots + \tilde{f}_m(x)$$

   where $(\pi_1, \cdots, \pi_m) \leftarrow X$ is drawn uniformly at random, and $\tilde{f}_i : X \to \mathbb{Z}_n$ are also uniformly random (subject to the above condition). Here $+$ symbols denote group operations over the respective groups. For $i \in [m]$, let $\rho_i = (\pi_i, \tilde{f}_i)$, where $\tilde{f}_i$ is represented as a function table, in $\mathbb{Z}_n^{(n_1, \cdots, n_m)}$. $\mathrm{Gen}(f)$ outputs $(\rho_1, \cdots, \rho_m)$.
2. **Encoding:** If $\rho_i = (\pi_i, \tilde{f}_i)$, then $\mathrm{Enc}_i(x_i, \rho_i) = (\tilde{x}_i, g_i)$ where $\tilde{x}_i = x_i + \pi_i$, and $g_i : \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n-1} \times \mathbb{Z}_{n+1} \times \cdots \times \mathbb{Z}_{n_m} \to \mathbb{Z}_n$ is such that

$$g_i(y_1, \cdots, y_{i-1}, y_{i+1}, \cdots, y_m) = \tilde{f}_i(y_1, \cdots, y_{i-1}, \tilde{x}_i, y_{i+1}, \cdots, y_m).$$

3. **Computing the result:** $\mathrm{Dec}(\{(\tilde{x}_i, g_i)\}_{i=1}^m)$ outputs $\sum_{i=1}^m g_i(\tilde{x}_1, \cdots, \tilde{x}_{i-1}, \tilde{x}_{i+1}, \cdots, \tilde{x_m})$.

A.1 Analysis of the NIMPC Potocol

Below, let P_i represent the i^{th} input party (running the algorithm Enc_i) and let P_0 be the aggregator (running the algorithm Dec).

Communication complexity: Let $l = \lceil \log n \rceil$ be the size of each output element and let $d = \min_{i \in [m]} n_i$. Each party P_i sends its share of the functionality table (g_i) projected onto its input to P_0. Each such g_i has at most $|X|/d$ cells each of size l. There are m such parties. Thus the communication complexity of our protocol is at most $\frac{|X| \cdot m \cdot l}{d}$ bits.

This improves over the communication complexity of the general protocols presented in [BGI+14, OY16]. We remark that a version of our protocol for the case of $n_i = 2$ for all $i \in [m]$ appears in [HIJ+16], but the improvement is more marked for larger input domains. Specifically, the protocol presented in [OY16] has a communication complexity of $|X| \cdot m \cdot l \cdot \log^2(d+1)$. Thus we obtain a $d \cdot \log^2(d+1)$ reduction in communication complexity. When $n_i = N$ for all $i \in [m]$, the communication complexity of our protocol is $N^{m-1} \cdot m \cdot l$ bits which is a factor of $N \log^2 N$ improvement over that of [OY16] and an N^3 over [BGI+14]. For a small number of parties (say, $m = 5$) and a moderately small input space (say, $N = 16$) our protocol is concretely quite efficient (for a boolean function, we need 320 KiB communication, compared to 80 MiB for [OY16] and 1.25 GiB for [BGI+14]).

Correctness: P_0 needs to output $f(x_1, \cdots, x_m)$. Let $\pi_i, \tilde{f}$, and $\tilde{f}_i$ be as generated by Gen(f). Also let $\tilde{x}_i = x_i + \pi_i$ (as defined by P_i). Then

$$\begin{aligned} f(x_1, \cdots, x_m) &= \tilde{f}(\tilde{x}_1, \cdots, \tilde{x}_m) \\ &= \sum_{i=1}^{m} \tilde{f}_i(\tilde{x}_1, \cdots, \tilde{x}_m) \\ &= \sum_{i=1}^{m} g_i(\tilde{x}_1, \cdots, \tilde{x}_{i-1}, \tilde{x}_{i+1}, \cdots, \tilde{x}_m) \end{aligned}$$

which is actually output by the aggregator. This shows that the protocol is perfectly correct.

Security: First consider the case when all input parties are honest, i.e., $T = \emptyset$. In this case the adversarial aggregator's view is $M(x, r) := \{(\tilde{x}_i, g_i) \mid i \in [m]\}$, where r denotes the random choices made by the protocol. Given two values $z, z' \in X$ such that $f(z) = f(z')$, we define a bijection $\Phi_{(z,z')}$ from the set of the random choices r of Gen(f) to itself, such that $M(z, r)$ to $M(z', \Phi_{(z,z')}(r))$.

Note that the randomness r can be identified with $\pi = (\pi_1, \cdots, \pi_m)$ and $\tilde{f}_1, \cdots, \tilde{f}_m$, subject to $f(x - \pi) = \sum_i \tilde{f}_i(x)$ for each $x = (x_1, \cdots, x_m)$). Given such an r, we define $\Phi_{(z,z')}(r)$ to consist of $\pi' = (\pi'_1, \cdots, \pi'_m)$ and $(\tilde{f}'_1, \cdots, \tilde{f}'_m)$ as follows.

$$\begin{aligned} \pi'_i &= \pi_i + z_i - z'_i && \text{for } i \in [m] \\ \tilde{f}'_i(x) &= \begin{cases} f(x - \pi') - \sum_{j \neq i} \tilde{f}_j(x) & \text{if } x_i \neq \tilde{z}_i \text{ and } \forall \ell < i,\ x_\ell = \tilde{z}_\ell \\ \tilde{f}_i(x) & \text{otherwise,} \end{cases} && \text{for } x \in X, i \in [m]. \end{aligned}$$

where $\tilde{z} := z + \pi = z' + \pi'$.

Firstly, note that $\tilde{f}'_i$ defined as above satisfy the condition that $f(x - \pi') = \sum_j \tilde{f}'_j(x)$ for all x: For $x \neq \tilde{z}$, there is exactly one coordinate i such that $x_i \neq \tilde{z}_i$ and $x_\ell = \tilde{z}_\ell$, so that $\tilde{f}'_i(x) = f(x - \pi') - \sum_{j \neq i} \tilde{f}_j(x) = f(x - \pi') - \sum_{j \neq i} \tilde{f}'_j(x)$. For $x = \tilde{z}$ we have $\sum_j \tilde{f}'_j(\tilde{z}) = \sum_j \tilde{f}_j(\tilde{z}) = f(\tilde{z} - \pi) = f(z) = f(z') = f(\tilde{z} - \pi')$. So, $\Phi_{(z,z')}$ maps valid choices of $(\pi, \tilde{f}_1, \cdots, \tilde{f}_m)$ when the input is z, to valid choices $(\pi', \tilde{f}'_1, \cdots, \tilde{f}'_m)$ when the input is z'.

Next, we argue that $M(z, r) = M(z, \Phi_{(z,z')}(r))$. Firstly, note that for all i, $\tilde{z}_i := z_i + \pi_i = z'_i + \pi'_i$. Also, for each i, g_i consists of $\tilde{f}_i$ evaluated on all inputs x with $x_i = \tilde{z}_i$. But at all such points, $\tilde{f}'_i(x) = \tilde{f}_i(x)$. So g'_i that is part of $M(z, \Phi_{(z,z')}(r))$ equals to g_i.

Finally, we observe that $\Phi_{(z,z')}$ is indeed a permutation over random choices. Indeed, $\Phi_{(z',z)}$ is the inverse of $\Phi_{(z,z')}$. To see this, note that the definition of π' and $\tilde{f}'_i$ are such that

$$\begin{aligned} \pi' - \pi &= z - z' \\ \tilde{f}'_i(x) - \tilde{f}_i(x) &= \begin{cases} f(x + z' - \tilde{z}) - f(x + z - \tilde{z}) & \text{if } x_i \neq \tilde{z}_i \text{ and } \forall \ell < i,\ x_\ell = \tilde{z}_\ell \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

That is, $\Phi_{(z,z')}(r) - r = \alpha(z, z')$ for some function α such that $\alpha(z, z') + \alpha(z', z) = 0$. which shows that $\Phi_{(z,z')}(\Phi_{(z,z')}(r)) = r$.

The above proof of security extends to a version of the protocol where t instances of the original protocol are used to evaluate t different functions (with the same input domains) $f^{(1)}, \cdots, f^{(t)}$, on the same input x, where the same π is used for all executions (but $\tilde{f}_i^{(k)}$ are sampled independently for

each k). This is because, $\Phi_{(z,z')}$ maps π in all instances to the same π' in all instances (independent of the function and the other randomness).

For the general case, suppose the adversary corrupts P_0 and P_i for $i \in T$ such that $T \subseteq [m]$ is not empty. Note that the protocol could be seen as parallel executions of the above protocol involving only input parties in $\overline{T}$, for functions $\hat{f}_y$ obtained by restricting f for each input $y \in X_T$ of the corrupt parties, using the same values π_i across all executions. Note that in all the parallel executions, parties in $\overline{T}$ use the same input. Hence, by the above observation, this protocol securely evaluates $\hat{f}_y$ for all $y \in X_T$, or equivalently, the residual function of the input of the parties in $\overline{T}$. This shows that the protocol is secure as an NIMPC protocol.

B A UC Secure Protocol for Latin Square Functions

Lemma 10. *Every $(n, 2)$-CCPS functionality has a UC secure protocol.*

This result follows from a result in [PR08b], which shows that a 3-party functionality is UC securely realizable iff every 2-party functionality obtained by partitioning the 3 parties into 2 parts yields a "splittable" function. With an $(n, 2)$-CPSS this property can be verified. Nevertheless, for the sake of completeness, we present an alternate simpler proof of this lemma, leveraging the compiler from Theorem 8.

Proof. Let $f = (X_1, X_2)$ be an $(n, 2)$-CCPS functionality (i.e., a Latin Square functionality), with $f(\pi_1, \pi_2) = \pi_1(\pi_2(1))$. We define a new (non-CPS) function $f^* : S_n \times X_2 \to [n]$, again defined as $f^*(\pi_1, \pi_2) = \pi_1(\pi_2(1))$. We also consider the following (insecure) protocol Π for computing f^* (π_1 and π_2 denoting the inputs of the two parties P_1 and P_2):

- P_1 sends a random permutation $\sigma \leftarrow S_n$ to P_2, and sends the permutation $\pi_1 \circ \sigma^{-1}$ to P_0.
- P_2 sends $\sigma(\pi_2(1))$ to P_0.
- P_0 receives a permutation $\rho \in S_n$ from P_1 and a value $x \in [n]$ from P_2. It outputs $\rho(x)$.

This protocol is insecure for f^* when the set $\{P_0, P_2\}$ is (passively or actively) corrupt: in the protocol, together they learn π_1 exactly, where as in the ideal world, they can only learn $\pi_1(a)$ for some $a \in [n]$, and since P_1's input domain is all of S_n, this does not let them learn π_1 exactly. However, Π *is a UC secure protocol for f^*, for all other corruption patterns* (i.e., when at most one of P_0 and P_2 is corrupt):

- When all parties are honest, the protocol produces the correct output because $\rho(x) = \pi_1 \circ \sigma^{-1} \circ \sigma \circ \pi_2(1) = \pi_1(\pi_2(1)))$, and hence simulation is trivial.
- When P_1 alone is corrupt, the simulator can extract a valid input for P_1 (in S_n) from the messages it sends to (simulated) P_0 and P_2, as $\rho \circ \sigma$.
- When P_2 alone is corrupt, any message in $[n]$ that it sends to (simulated) P_0 uniquely maps to a valid input, given the permutation σ it received from (simulated) P_1.
- When P_0 alone is corrupt its view – which consists of a random pair $(\rho \in S_n, x \in [n])$ such that $\rho(x)$ is the output – can be perfectly simulated from the output.
- When P_1, P_2 are corrupt, any (ρ, x) they send to P_0 corresponds to a valid input combination (in fact, n combinations) of P_1, P_2 that results in the output $\rho(x)$.
- When P_0, P_1 are corrupt, in the ideal world they can learn P_2's input π_2 from $(\pi_1, \pi_1(\pi_2(1))$ because $\pi_2(1)$ uniquely determines $\pi_2 \in X_2$.

- Simulation is trivial when all 3 are corrupt.

Now, the compiler in Theorem 8 in fact retains its guarantees for any adversary structure (i.e., set of parties who can be corrupted). For $D = X_1 \times X_2$, $f = f_D^*$ and f is a CPS. Thus the protocol Π_D obtained by applying the compiler of Theorem 8 to Π is in fact UC secure for f for any adversary structure in which at most one of P_0 and P_2 is corrupt.

Next we claim that Π_D is in fact a UC secure protocol for f even when $\{P_0, P_2\}$ are both corrupt. This is because, in this case, they can learn π_1 even in the ideal world for f (since f is a CPS), and a perfect simulation is possible. □

C Examples

Here we collect a few concrete examples used earlier in the paper and make additional comments on them.

- **Example of a Latin Hypercube that is not a CPS.** Consider the 3-dimensional, 4-ary Latin hypercube functionality given by $f(x_1, x_2, x_3) = (-1)^{x_2+x_3}(x_1 - x_3) + x_2$ where all operations are modulo 4 (here, instead of 0 we shall write 4, so that the output alphabet is [4], to be consistent with our convention). It can be verified that f is a Latin hypercube function, and, $f(1,1,1) \neq f(2,2,1)$ but $f(1,1,4) = f(2,2,4)$. This contradicts a requirement for f to be a CPS functionality, namely that there should be a function (permutation) π such that for all x, y, $f(x, y, 4) = \pi(f(x, y, 1))$.
- **Example of a Latin Square CPS that is not a CPSS:** Every Latin square is a CCPS functionality. We show one such function which is not a CPSS, showing that $m > 2$ is required in Lemma 5.
  Consider the following Latin square:

	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5
π_1	1	2	3	4	5
π_2	2	3	1	5	4
π_3	3	5	4	1	2
π_4	4	1	5	2	3
π_5	5	4	2	3	1

  Here ρ_1 and π_1 correspond to the identity permutations. We can see that ρ_2, ρ_3, ρ_4 do not have their inverses in the set $\{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$ (nor do π_2, π_3, π_4 in the set $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$).
  Further, this function cannot be *embedded* into a larger CPSS either. To see this, first we note that (ignoring relabeling) if $\rho_2(\alpha) = \rho_3^{-1}(\alpha)$ for some α, then $\rho_2 = \rho_3^{-1}$: being a CPSS would require that ρ_3^{-1} is one of the permutations in the input set, and then being a CPS with two input permutations ρ_2 and ρ_3^{-1} which coincide on some α would require that $\rho_2 = \rho_3^{-1}$. However, we have $\rho_2(1) = 2 = \rho_3^{-1}(1)$, but $\rho_2(2) = 3 \neq \rho_3^{-1}(2) = 5$.
- **Concrete Challenges.** The following functionality f is the same as the function in the proof of Theorem 1 for the case of $m = 3$, written out explicitly (using output alphabet [5] and input alphabet $\{0, 1\}$ for each party). There it was shown that $f \notin$ **CPSS**, and hence none of our UNIMPC* protocols can be used to compute this function securely. On the other hand, from Theorem 4, we know that $f \in$ **UNIMPC**. As such, we present f as a candidate function for

separating **UNIMPC** from **UNIMPC***, or alternately, as a challenge for devising new UNIMPC* protocols.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	$f(\overline{x_1}, \overline{x_2}, \overline{x_3})$
0	0	0	1	5
1	0	0	2	2
0	1	0	3	3
0	0	1	4	4

Similarly, considering the case $m = 4$ in the construction from the proof of Theorem 1, we get an explicit challenge for a UNIMPC protocol. Due to a result in [HIKR18] for 4-input functions, this has an MPC protocol; a challenge for MPC can be constructed by using $m = 5$.