Secure Multi Party Computation

Bachelors Thesis Project

Bachelors of Technology
in
Computer Science and Engineering

by

Navneet Agarwal & Sanat Anand
(140100090 & 140040005)

under the guidance of

Prof. Manoj Prabhakaran

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

May, 2018

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 (2001), 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 give a necessary condition and a slightly stronger sufficient condition for f to admit a secure protocol. Both the conditions are stated in terms of an algebraic structure we introduce called "Commuting Permutations Systems" (CPS), which may be of independent combinatorial interest.

When our sufficiency condition is met, we obtain a perfectly secure protocol with minimal interaction, that fits the model of Non-Interactive MPC or NIMPC (Beimel et al., 2014), but without the need for a trusted party to generate correlated randomness. We define Unassisted Non-Interactive MPC (UNIMPC) to capture this variant. We also present an NIMPC protocol for all functionalities, which is simpler and more efficient than the one given by Beimel et al.

In addition to this, we present new security definition namely "Strong security" and give a necessary condition and sufficient condition for the same. We also present a UC-security compiler which can used to create secure protocols for a UC-secure functionality with an increased domain.

Declaration

We declare that this written submission represents our ideas in our own words and where other's ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: 09^{th} May, 2018

Place: IIT Bombay, Mumbai

Navneet Agarwal & Sanat Anand

Roll No: 140100090 & 140040005

Acknowledgements

This report is an outcome of our work under the guidance of Prof. Manoj Prabhakaran on Secure Multi Party Computation. We are thankful to the people who have been instrumental in helping us out throughout this Research project. First and foremost, we would like to express our sincere gratitude towards our supervisor Prof. Manoj Prabhakaran for his guidance. Our research is being driven by his vision and support. It was immensely helpful in gaining the understanding required for writing this report. Major parts of our report (Chapters 1-7) have been borrowed from our research paper co-authored with him. We thank Fred Douglas for his work on disseminated functionalities which we have included in Chapter 9.

We would also like to thank Sohum Dhar for brilliant ideas, for healthy and useful discussions on the topic and Saurabh Garg for helping us format the BTP report.

Contents

Chapter 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 [3, 12, 14]. 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 protocols?*

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 [9]. Chor and Ishai pointed out the difficulty of this problem, by showing that one cannot directly reduce characterization of k-party functionalities to that in a setting with fewer parties in an obvious fashion [4]. Since then, very little progress has been made on this problem.

The main motivation of this work is to advance the state-of-the-art on this front. Towards this, we focus on an important class of functionalities called *aggregating functionalities*, in which there are a set of "input parties" and a single "output party." We ask (and partially answer) the following question:

> *Which aggregating functionalities have MPC protocols that are information-theoretically secure against passive corruption?*

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 [6]. This model – sometimes refered to as the

Private Simultaneous Messages (PSM) model – served as a precursor of important concepts like randomized encodings [7] 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. [1], which is closer to standard MPC in terms of the security requirements. In both these models the severe restriction on the communication pattern often leads to simple and elegant protocols. Indeed, for specialized functions (like "Remote-OT" and AND) the original protocols developed in the PSM model [6] can also be shown to be optimal (or very nearly so) in terms of communication and randomness complexity [5]. Similarly, Beimel et al. discovered several elegant NIMPC protocols for special classes of functions [1]. However, these protocols do not directly translate to MPC protocols *as these models include a trusted party* which sends correlated random variables to the parties in a pre-processing phase. The term aggregating functionality was coined in [10].

In studying aggregating functionalities, we introduce the notion of *Unassisted NIMPC* (or UNIMPC for short) which helps us state our results more sharply, and also is a model of independent interest. UNIMPC, as the name suggests, removes the assistance from the trusted party in NIMPC and instead requires that the parties securely compute the correlated randomness by themselves, in a (one-round) offline phase. Like PSM and NIMPC, UNIMPC is also a minimalistic model for secure computation, in terms of the communication pattern allowed (after the input is received). But unlike PSM and NIMPC, which allow trusted parties, *UNIMPC retains the standard security model of MPC*, allowing corruption of any set of parties. While 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), UNIMPC could be seen as a common denominator of these two secure computation models.

> *A UNIMPC protocol can be directly interpreted as an MPC protocol as well as an NIMPC protocol.*

Our main positive result will in fact give a UNIMPC protocol for a class of functionalities called CPSS (see below). This result is contrasted by a negative result showing that the only functionalities that have MPC protocols belong to a class called CPS. The two classes CPS and CPSS have algebraic definitions that closely resemble each other. In both these classes *the inputs of the parties are permutations* and the output is obtained by applying the permutations of all the parties successively to a fixed element (say 1). CPS stands for a *Commuting Permutations System*, in which the order of the application of the permutations among the parties is not important. (Commuting is guaranteed only when the composed functions are applied to the fixed element 1. Also, two permutations in the same party's input set need not commute even in this limited sense.) CPSS stands for *Commuting Permutation Subgroups Systems*, which requires the set of permuations that form the input domain of each party should be a subgroup of the symmetric group.

Our results can be summarized as follows. For any number of parties,

$$\text{CPSS} \Rightarrow \text{UNIMPC}^* \Rightarrow \text{UNIMPC} \Rightarrow \text{MPC} \Rightarrow \text{CPS}.$$

Here 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.)

Note that we leave an intriguing gap between the necessary and sufficient conditions. In particular we leave open the possibility that the set of functionalities with UNIMPC protocols is a strict subset of the set of aggregating functionalities with MPC protocols, and is a strict superset of aggregating functionalities with UNIMPC* protocols. However, these differences disappear for small number of parties: When the number of input parties is 2, we show that UNIMPC* $\Leftrightarrow$ CPS, and when the number of input parties is 3, UNIMPC $\Leftrightarrow$ CPS.

Our results could be seen as a step towards fully characterizing the functionalities with information-theoretic MPC protocols in various security models. For instance, for characterizing functionalities with UC secure protocols, aggregating functionalities remain the only class to be understood [10], and the sub-classes of aggregating functionalities identified in this work can serve as a starting point for understanding UC security. Similarly, the problem of characterizing symmetric functions (when all parties get the same output) as considered in [4] is still unsolved, but our positive results do present new possibilities there (because a passive-secure MPC protocol for an aggregating functionality can be readily converted into one for a symmetric functionality computing the same function).

Chapter 2

Prerequisites

In this chapter, we will discuss some prerequisites and definitions necessary to understand our results.

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 $[n]$.

Notation. *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 $\prod_{i \in [t]} \rho_i$).*

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

We borrow some notation and definitions of NIMPC and PSM from [1].

The framework of universal composability (UC) is a general-purpose model for the analysis of cryptographic protocols. It guarantees very strong security properties. Protocols remain secure even if arbitrarily composed with other instances of the same or other protocols. [13]

Definition 1 (UC Security)**.** We define the "ideal world" as an analogue to real world containing a trusted third party which executes the functionality. An adversary($\mathcal{A}$) in the real world which corrupts one or more parties is in full control of their behaviour. In the ideal world a special party called the simulator($\mathcal{S}$) controls the behaviour of all the parties corrupted by the adversary in the real world. The environment($\mathcal{E}$) gives the inputs and receives the outputs from the parties and interacts arbitrarily with the adversary. Once the protocol execution is complete the environment outputs a bit (b) according to its state. We say that a protocol (Π) is said to be UC secure for a

given functionality if,

$$\forall \mathcal{A} \exists \mathcal{S} : \forall \mathcal{E}, b \cong b'$$

where b' is the bit output by the $\mathcal{E}$ in the real world.

Definition 2 (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 $\mathcal{H}$ be a family of functions $h : X \to \Omega$. A non-interactive secure multiparty computation (NIMPC) protocol for $\mathcal{H}$ is a triplet $\Pi = (Gen, Enc, Dec)$ where

- $Gen : \mathcal{H} \to R_1 \times \cdots \times R_m$ is a randomized function.
- Enc is an m-tuple of deterministic functions $(Enc_1, \cdots, Enc_m)$, where $Enc_i : X_i \times R_i \to M_i$,
- 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 $h \in \mathcal{H}$,

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

  where $Enc(x, r) = (Enc_1(x_1, r_1), \cdots, Enc_m(x_m, r_m))$.

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

Definition 3 (NIMPC Security)**.** We say that an NIMPC protocol Π for $h : X \to \Omega$ is T-robust for $T \subseteq [m]$, if there exists a randomized function Sim (a simulator) such that $\forall x_{\overline{T}} \in X_{\overline{T}}$, we have $Sim_T(h_{\overline{T}, x_{\overline{T}}}) = (M_{\overline{T}}, R_T)$, where R and M are the joint randomness and messages defined by $R \leftarrow Gen(h)$ and $M_i \leftarrow Enc_i(x_i, R_i)$. Π is said to be *secure* if it is T-robust for all $T \subseteq [m]$.

A Private Simultaneous Message (PSM) protocol [6] is simply a $\emptyset$-robust NIMPC. Next, we present our definition of Unassisted NIMPC.

We adapt the definition of an *aggregating functionality* and *disseminating functionality* from [10]. The original definition in [10] allows all the parties to have outputs, but requires that for each party other than the aggregator, its output is a function only of its own input. Such a function is "isomorphic" to an aggregating functionality as we define below. In our UNIMPC model, protocols are defined only for aggregating functionalities as defined here.

Definition 4 (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.

Similar to above, the defintion of disseminating functionality in [10] allows every party to have an input but for every party other than the disseminator, the input must only affect its local output. Also the disseminator may have an output. Such a function is "isomorphic" to a disseminated functionality as defined here.

Definition 5 (Disseminated Functionality)**.** An $(m+1)$ party *Disseminated functionality* accepts input $x \in X$ from P_0 (also called the disseminator) and sends $f(x)_i \in Y_i$ to P_i for every $i \in [m]$, where $f : X \to Y_1 \times \cdots \times Y_m$ 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 6. Two aggregating functionalities $f : A_1 \times \cdots \times A_m \to A_0$ and $g : B_1 \times \cdots \times B_m \to B_0$ are said to be *isomorphic* to each other if there are bijections $\phi_i : A_i \to B_i$ for $i = 1, \cdots, m$, and a bijection $\phi_0 : \text{Image}(f) \to \text{Image}(g)$ s.t. for all $(a_1, \cdots, a_m) \in A_1 \times \cdots \times A_m$, $f(a_1, \cdots, a_m) = \phi_0(g(\phi_1(a_1), \cdots, \phi_m(a_m)))$.

Definition 7 (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 $x \in X$:

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

where $View_x(\{P_i | i \in T\})$ 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 [8].

Lemma 8 (2-Party MPC with one-sided output [8])**.** *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 $\exists y \in Y, f(x, y) = f(x', y) \Rightarrow \forall y \in Y, f(x, y) = f(x', y)$.*

Chapter 3

New Models

In this chapter we define UNIMPC and UNIMPC*, which are models of secure computation, as well as combinatorial objects CPS and CPSS.

Definition 9 (UNIMPC)**.** We define an Unassisted Non-Interactive Secure Multi-party Computation (UNIMPC) protocol Π for functionality $f : X \to \Omega$ as $\Pi = (\mathcal{R}, Enc, Dec)$ where:

- $\mathcal{R}$ is an m-party MPC protocol secure against passive corruption for generating randomness $(r_1, \cdots, r_m) \in R_1 \times \cdots \times R_m$.
- Enc is an m-tuple of deterministic functions $(Enc_1, \cdots, Enc_m)$ where $Enc_i : X_i \times R_i \to M_i$.
- Dec : $M_1 \times \cdots \times M_m \to \Omega$ is a deterministic function satisfying the following correctness requirement: for any $x = (x_1, \cdots, x_m) \in X$

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

  where $Enc(x, r) = (Enc_1(x_1, r_1), \cdots, Enc_m(x_m, r_m))$.

It is a two-phase MPC protocol (secure against passive corruption) where:

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

Security: A UNIMPC protocol is said to be secure if it is a passive secure MPC protocol for f (as in Definition 7). More specifically, a UNIMPC protocol Π for $f : X \to \Omega$ is said to be T-robust (for

$T \subseteq [m]$) if there exists a simulator S s.t. for any $x \in X$:

$$View_x(\{P_i | i \in T\} \cup \{P_0\}) \cong S(x_T, f(x))$$

where $View_x(\cdot)$ represents the view of a given set of parties in an execution of Π with input x.[1]

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

Definition 10 (UNIMPC*)**.** We define an Unassisted Non-Interactive Secure Multi-party Computation with Non-Interactive Preprocessing (UNIMPC*) protocol Π for some functionality $f : X \to \Omega$ as a UNIMPC protocol $\Pi = (\mathcal{R}, Enc, Dec)$ for f where $\mathcal{R}$ consists of a single round.

Definition 11. 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 $\rho \in S_m$, $\pi_1 \circ \cdots \circ \pi_m(1) = \pi_{\rho(1)} \circ \cdots \circ \pi_{\rho(m)}(1)$.[2]

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 12. 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 functionality) if $(X_1, \cdots, X_m)$ is an (n, m)-CPS (resp. (n, m)-CPSS) and for all $(\pi_1, \cdots, \pi_m) \in X_1 \times \cdots \times X_m$, $f(\pi_1, \cdots, \pi_m) = (\prod_{i \in [m]} \pi_i)(1)$.

[1] Note that in defining T-robustness we include only the case when the set of corrupt parties includes the aggregator. Security when the aggregator is honest is guaranteed from the structure of the UNIMPC protocol, since the view of the adversary in this case is derived completely from the offline phase.

[2] 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.

Chapter 4

Necessary Condition for (UNI)MPC

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 13. *If an aggregating functionality has an information-theoretically secure MPC protocol against semi-honest adversaries, then it is isomorphic to 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 isomorphic function 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 8, 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): $\forall \mathbf{x}, \mathbf{x}' \in X_1 \times \cdots \times X_m$,

$$f(\mathbf{x}) = f(\mathbf{x}') \text{ and } x_i = x_i' \;\Rightarrow\; \forall \ell \in X_i, f(\mathbf{x}[i : \ell]) = f(\mathbf{x}'[i : \ell]). \tag{4.1}$$

We use this to prove the following claim.

Claim 1. *There exist permutations $\pi_\ell^{(i)}$ for $i \in [m]$ and $\ell \in X_i$ such that for $\mathbf{x} \in X_1 \times \cdots \times X_m$ with $x_i = 1$,*

$$\pi_\ell^{(i)}(f(\mathbf{x})) = f(\mathbf{x}[i : \ell]). \tag{4.2}$$

Proof. By Equation 4.1, one can define a function $\pi_\ell^{(i)}$ to satisfy the condition in the claim. Note that even though there could be multiple $\mathbf{x}$ with $x_i = 1$ and the same value for $f(\mathbf{x})$, Equation 4.1 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 4.1, 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 4.2, $\pi_{\ell_t}^{(i_t)} \circ \cdots \circ \pi_{\ell_1}^{(i_1)}(f(\mathbf{x})) = f(\mathbf{x}[i_1 : \ell_1] \cdots [i_t : \ell_t])$. 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 have substituted $f(\mathbf{x}) = 1$. This concludes the proof that f is isomorphic to the CPS functionality with input domains $\hat{X}_i = \{\pi_\ell^{(i)} | \ell \in [n_i]\}$. □

4.1 Necessity of CPS for 1-Robust UNIMPC and Implications for NIMPC

Since every secure UNIMPC protocol is a secure MPC protocol, Theorem 13 applies to UNIMPC as well. But we point out that in fact the theorem extends to UNIMPC in a stronger manner than it holds for MPC. If we restrict the number of corrupt parties to be at most $m/2$, then every $m + 1$ 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 4.1 was derived in the proof of Theorem 13 (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 8 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-robust for every T of the form $\{0, i\}$ ($i \in [m]$), then Lemma 8 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 [1], and we cannot require all such functions to be CPS. But note that NIMPC turned out to be possible for all functions only 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). Here, we note that the above argument extends further to the NIMPC setting where there is a trusted party which sends correlated randomness to all the parties: 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).

Chapter 5

UNIMPC Protocols (Sufficiency)

In this section we present our feasibility results for UNIMPC* and UNIMPC, namely Theorem 14 and Theorem 15.

Theorem 14. *Any function isomorphic to a function embeddable in a CPSS function has a UNIMPC* protocol with perfect security.*

To prove Theorem 14 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

$$\sigma_{1j} \circ \sigma_{2j} \circ \cdots \circ \sigma_{mj}(1) = 1. \tag{5.1}$$

   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.

In Section 5.1 we prove that this protocol is a perfectly secure UNIMPC* protocol for f.

Theorem 15. *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. 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 [6], except that all the randomness is sampled by P_1 and sent to P_2.[1] 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 (Gen, Enc, Dec) for the given CPS functionality (see Section 7), 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 [2]).

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 $z_i := Enc(x_i, r_i)$ to P_0 and P_0 outputs $Dec(z_1, \cdots, z_m)$.

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)

[1] 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.

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. □

5.1 UNIMPC* Protocol Security for CPSS Functions

We prove that the protocol for a CPSS functionality used in the proof of Theorem 14 is perfectly correct and perfectly secure. We use the same notation below as in the that section.

Perfect Correctness: The correctness is based on the following lemma.

Lemma 16. *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*

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

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, we have

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

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

Now, the output of P_0 is $\prod_{i=0}^{m}\sigma_{i,0}(1)$. By Equation 5.1 (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 5.1 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 the above claim, 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: Security when the aggregator is honest is easy to see: the messages received by the corrupt parties from the honest parties (of the form σ_{ij} where $j \neq 0$) are from a fixed distribution independent of the inputs of the honest parties. Combined with perfect correctness, this implies perfect semi-honest security.

Also, perfect semi-honest security trivially holds when all the 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 lemma completes the proof of security.

Lemma 17. *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

$$(\prod_{i\in[m]} \sigma_{ij})(1) = 1 \qquad \forall j \in [m] \tag{5.2}$$

$$(\prod_{i\in[m]} \beta_i)(1) = (\prod_{j=h+1}^{m} \prod_{i\in[m]} \sigma_{ij})(1) = 1. \tag{5.3}$$

where the first equality in (5.3) is obtained by applying Lemma 16, and the second equality is obtained by applying the validity condition (5.2) 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 $\beta := \prod_{i\in[m]} \beta_i$):

$$\begin{aligned}
&\prod_{i\in[m]} \pi'_i(1) = \prod_{i\in[m]} \pi_i(1) \\
&\quad\Rightarrow (\prod_{i\in[m]} \alpha_i \circ \sigma'_{ih} \circ \beta_i)(1) = (\prod_{i\in[m]} \alpha_i \circ \sigma_{ih} \circ \beta_i)(1) \\
&\quad\Rightarrow \alpha \circ (\prod_{i\in[m]} \sigma'_{ih}) \circ \beta(1) = \alpha \circ (\prod_{i\in[m]} \sigma_{ih}) \circ \beta(1) && \text{by Lemma 16} \\
&\quad\Rightarrow (\prod_{i\in[m]} \sigma'_{ih}) \circ \beta(1) = (\prod_{i\in[m]} \sigma_{ih}) \circ \beta(1) && \text{as } \alpha \text{ is a permutation} \\
&\quad\Rightarrow (\prod_{i\in[m]} \sigma'_{ih})(1) = (\prod_{i\in[m]} \sigma_{ih})(1) = 1 && \text{by (5.3) and (5.2).}
\end{aligned}$$

□

Chapter 6

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 $[n]$ (see Definition 19 for a succinct definition). We can associate an m-input functionality with a Latin hypercube, which maps the index vector to the corresponding entry in the hypercube.

In the case of $m = 2$, an n-ary Latin square functionality f is always (isomorphic to) an $(n, 2)$-CPS (X_1, X_2).[1] However, this is not true in higher dimensions. So not all Latin hypercube functions can have MPC protocols. From our results above, we obtain an exact characterization of all Latin hypercube functionalities that have UNIMPC* (or MPC) protocols. Recall that by Theorem 13 only CPS functionalities can have UNIMPC* (or even MPC) protocols. We show that *all Latin hypercube functionalities that are CPS functionalities indeed have UNIMPC* protocols*.

Theorem 18. *A Latin hypercube functionality has a UNIMPC* protocol if and only if it is a CPS functionality.*

Proof. The "only if" direction follows from Theorem 13. We need to argue that every Latin hypercube functionality that is a CPS functionality has a UNIMPC protocol. If the functionality has up to 4 parties, it follows from Theorem 15 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. Thus, as a CPS, it is complete in at least 3 dimensions. By Lemma 23, it is

[1] 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 an isomorphism relabeling the outputs of f so that $f(1,1) = 1$, this meets the definition of a CPS.

an FCPS, and hence by Lemma 21 it is embeddable in an FCPSS. Then, using Theorem 14, the functionality has a UNIMPC protocol. □

One can represent 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 19. $L \subseteq [n]^{m+1}$ 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.

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 (writing 4 instead of 0, to be consistent with the definitions). 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, $f(x, y, 4) = \pi(f(x, y, 1))$.

6.1 Complete CPS, FCPS and FCPSS

Definition 20. 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 21. *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. □

Definition 22. An (n, m)-CPS $(X_1, \cdots, X_m)$ is said to be *complete in dimension i* if $\{\pi(1) \mid \pi \in X_i\} = [n]$.

Lemma 23. *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). One can construct Latin squares which do not correspond to an FCPS (even though all of them are CPS).

Chapter 7

More Efficient General NIMPC Protocol

We give a simple construction of a perfectly secure NIMPC protocol for any function in the information theoretic setting, which is more efficient than the one given in [1].

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.) Let P_i represent the i^{th} input party, let $\mathcal{H}$ be the trusted party generating shared randomness and let P_0 be the aggregator.

1. **Randomness Generation:** $\mathcal{H}$ picks $\{\pi_1, \cdots, \pi_m\} \leftarrow X$ uniformly at random. and defines $\tilde{f}$ such that $\forall (x_1, \cdots, x_m) \in X$:

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

   Also, $\mathcal{H}$ defines uniformly random functions $\tilde{f}_i : X \to \mathbb{Z}_n$, for $i \in [m]$, such that $\forall x \in X$:

   $$\tilde{f}(x) = \tilde{f}_1(x) + \cdots + \tilde{f}_m(x)$$

   Above $+$ symbols denote group operations over the respective groups. $\mathcal{H}$ then sends $(\pi_i, \tilde{f}_i)$ to P_i for each $i \in [m]$ (where $\tilde{f}_i$ is represented as a function table, in $\mathbb{Z}_n^{(n_1, \cdots, n_m)}$).

2. **Interaction with the aggregator:** Each party P_i, with input x_i, computes $\tilde{x}_i = x_i + \pi_i$. Also P_i defines the function $g_i : \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n-1} \times \mathbb{Z}_{n+1} \times \mathbb{Z}_{n_m} \to \mathbb{Z}_n$ such that

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

   P_i sends $(\tilde{x}_i, g_i)$ to P_0.

3. **Computing the result:** P_0 outputs $\sum_{i=1}^{m} g_i(\tilde{x}_1, \cdots, \tilde{x}_{i-1}, \tilde{x}_{i+1}, \cdots, \tilde{x_m})$

We give a detailed proof of security in Section 7.1.

Communication complexity: Let l 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 $O(|X|/d)$ cells each of size l. There are m such parties. Thus the communication complexity of our protocol is $O(\frac{|X| \cdot m \cdot l}{d})$.

This improves over the communication complexity of the general protocol presented in [1]. Specifically, if we have $n_i = N$ for all $i \in [m]$, we obtain a $\Theta(N^3)$ factor reduction in the communication complexity ($\Theta(N^{m-1} \cdot m \cdot l)$ vs. $\Theta(N^{m+2} \cdot m \cdot l)$). As pointed out in [1], their general protocol is fairly practical when there is a moderate number of participants (say 20), each with a small input. We point out that even if each input is a single bit, our protocol gives an $8\times$ improvement, which would be practically significant.

7.1 Security Proof

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 $\mathcal{H}$. 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 $\mathcal{H}$ 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.

$$\pi'_i = \pi_i + z_i - z'_i \qquad \text{for } i \in [m]$$

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

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

$$\pi' - \pi = z - z'$$

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

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 is secure.

Chapter 8

UC Compiler for domain restriction

We propose a compiler which turns a UC secure protocol for inputs from a given domain space (Π) into a statistically UC secure protocol for a restricted domain space (Π').

8.1 Basic outline

We run the standard addition functionality F on random inputs for some N times. The second step is for the aggregator to choose a subset of values from [N] and get those values from the parties. So if a party has sent in too many corrupt inputs there is a very high chance that it will get caught in this very step. The aggregator then checks the consistency of the values received. If not consistent we can abort. Now we execute the AND functionality on the remaining subset values with the input as described above. Now with some finite probability all the random inputs in an execution matches the true inputs. This probability multiplied by the number of executions is the expected number of ands. If the corrupt party sends a very less number of ands = 1, then it will not be comparable to the expected number of ands and we can abort. And if its the case that the number of ands is large as sent by the corrupt party then we take an honest majority. Because the number of corrupt inputs is of the order $\Theta(log(poly(N)))$ and number of 1s is $\Theta(N)$. Hence we can take the honest majority. This compiler can be used independently for UC secure majority voting. It can also be used as an intermediate to create a compiler for certain semi honest protocols to UC secure protocols.

8.2 Protocol description

Let F represent the functionality to be realized and k be the security parameter. Let $\mathcal{E}$ be the given domain space and $\mathcal{D}$ be the restricted domain space. 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:** Run k copies of the UC secure protocol (Π) for functionality F with domain $\mathcal{E}$. Each honest party $P_i, i \in [m]$ chooses input uniformly at random and randomness as per the specified distribution thereof (from domain $\mathcal{D}$). Let u_{ij} and r_{ij} be the input and randomness respectively used by party P_i in the j^{th} execution and let v_j be its output.

2. **Opening:** P_0 chooses $S \subseteq [k], |S| = k/2$ and every party $P_i, i \in [m]$ sends $u_{ij}, r_{ij} \forall j \in S$ to P_0. Then, P_0 checks the consistency of all the inputs and randomness received by checking if $F(\{u_{ij}\}_{i\in[m]}) = v_j \forall j \in S$ with randomness set as $\{r_{ij}\}_{i\in[m]}$. It also confirms that each input is chosen from the appropriate domain space ($\mathcal{D}$). Otherwise P_0 Aborts.

3. **Tallying with actual inputs:** Run $k/2$ copies of the AND functionality with each index as $\bar{S} = [m] \setminus S$. Each honest party P_i sets the j^{th} AND input 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} AND be b_j. Also let $T = \{j : b_j = 1\}$.

4. **Computing the result:** P_0 outputs L if $|T| \geq \mathcal{G}/2$ where $\mathcal{G} = k/(2 * \prod_{i\in[m]} |X_i|)$ and $L = \max_{l\in[n]} |\{j \in T : v_j = l\}|$. Otherwise P_0 Aborts.

Correctness: Correctness is based on the following two lemmas:

Lemma 24. *If all parties P_i are honest, then $\mathbb{P}[Abort]_{\{u_{ij}\}\leftarrow X_i} = negl(k)$*

Proof. Abort is triggered by P_0 in precisely two cases: when the reported inputs and randomness do not match the obtained outputs in Step 2 or when $|T| < \mathcal{G}/2$ in Step 4. The first case is not applicable for honest parties as there'd never be any discrepancy in their reporting.

For the second case, we investigate $\mathbb{P}[|T| < \mathcal{G}/2]_{\{u_{ij}\}\leftarrow X_i}$. For the AND value to be 1 for the j^th execution, all the parties' inputs $\{u_{ij}\}_{i\in[m]}$ must match their actual inputs ($u_{ij} = x_i, \forall i \in [m]$). This occurs with probability $1/(\prod_{i\in[m]} |X_i|)$. As there are $k/2$ executions of the protocol, $\mathbb{E}[|T|] = \frac{k/2}{\prod_{i\in[m]} |X_i|} = \mathcal{G}$.

By the Multiplicative Chernoff Bound, for any $Y_1, \cdots, Y_k$ independent random variables in $\{0, 1\}$ with $Y = \sum Y_i$ and $\mu = \mathbb{E}[Y]$, for any $\delta > 0$, we know that $\mathbb{P}[X < (1-\delta)\mu] < (\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}})^{\mu}$.

We can substitute $\delta = 1/2$, $Y_j = b_j, \forall j \in \bar{S}$, $Y = \sum_{j \in \bar{S}} b_j = |T|$ and $\mu = \mathcal{G}$ to obtain the required result. □

Lemma 25. *If Abort is not triggered with an honest aggregator, then* $\mathbb{P}[Opt \neq F(x_1, \cdots, x_m)]_{\{u_{ij}\} \leftarrow X_i} = negl(k)$

Proof. We show that the any deviation of the adversary cannot affect the final output more than a negligible probability.

Case 1 : Too many corrupt inputs sent If c corrupt inputs are sent by the adversary in the executions of F, the probability of not being caught is negligible. The prob of not being caught = ${}^{k-c}C_{k/2} / {}^{k}C_{k/2}$. It can be proved using induction that the required probability is $\leq 1/2^c$. Therefore, if $c \geq \omega(log(poly(k)))$, then the probability of not being caught is negligible. Therefore, for the adversary to non-negligibly continue into the next step, $c = O(log(poly(k)))$

Case 2 : Too few corrupt inputs but too few ANDs as 1 The corrupt parties can potentially send very few corrupt inputs but send the bits of the AND as 1 for only those corrupt inputs. We show that this does not work as if it sends too few ANDs as 1, then the number of ANDs will be too low and the protocol will not continue into the next step. Expected Number of ANDs as 1 ($\mathcal{G}$) = $\frac{k}{2(2*d^m)}$. We have said that if the number of 1s obtained in AND is less that $0.5 * \mathcal{G}$, we will abort. Therefore, by the nature of design the corrupt parties must give atleast $\Omega(k)$ AND values as 1 to avoid being aborted.

Case 3 : Few corrupt inputs and several AND values as 1 This is the final case where the corrupt parties have given few corrupt inputs and have given ample number of 1s in AND. Here show that the majority taken by the aggregator will correspond to a corrupt value with only a negligible probability. We have seen that the number of 1s in the AND is $\Theta(k)$ and the number of corrupt inputs is just at max $\Theta(log(poly(k))$. Therefore, there are all but log many honest inputs whose values are to be considered in the majority. For the majority to correspond to one of the corrupt inputs, the number of corrupt inputs has to be substantially greater than the total number of inputs for which the AND is 1. We can see that if all the conditions up to this point are followed, the probability that the number of ANDs as 1 is less than twice the number of corrupt inputs is negligible thereby making it negligible for one of the corrupt functionality results to be output. □

Hence, by all-but-negligible probability, the output is guaranteed to be correct when obtained.

Security: When the aggregator is corrupt, then the only non-trivial thing for the adversary to do would be to learn something other than the honest parties' residual function. All communications between the aggregator and the parties is through the protocol Π and AND functionality which are assumed to be ideal. Thus the security of the functionalities guarantees the security of the new protocol (UC property).

When the aggregator is honest, we say that a set of adversarial parties ($P_A, A \subseteq [m]$) can *explain* their behaviour transcript (t_A) in some functionality execution (F_j) if $\exists u_{ij} \in \mathcal{D}, r_{ij}, i \in A | F_j(u_{ij}) = F_j(t_A)$ under the choice of randomness r_{ij}.

As the original protocol is perfectly secure, under any choice of input and randomness, when the protocol steps are followed as per rule it remains secure. The compiled version too will render every *explainable* protocol execution secure as

1. Abort probability is negligible if all parties are honest:
2. If the protocol doesn't abort, it has only a negligible probability of computing the wrong output

Chapter 9

Strong security for general MPC

In this chapter we define the notion of "strong" security. We also give a necessary condition and a separate sufficient condition for general MPC functionalities having strongly secure protocol. We have a small gap between necessary and sufficient conditions which can be bridged in future works.

Definition 26. We say that a protocol (Π) is strongly secure for a functionality ($\mathcal{F}$) if it is both semi-honest secure and UC secure for $\mathcal{F}$.

Composition Theorem: Note that strong security follows the composition property because both semi-honest security and UC security are composable.

Definition 27. An (n, m)-CPSS $(X_1, \cdots, X_m)$ is said to be a complete CPSS (CCPSS) if it is *complete in dimension i $\forall i \in [m]$ (as in Definition 22).*

Lemma 28. *The UNIMPC protocol (described in Section 5) is UC secure for complete CPSS*

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 $\forall \mathcal{A}_R$, $\exists \mathcal{A}_H : View_{x_{\overline{T}}}(\{P_i | i \in \overline{T}\})$ 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 $\pi_i = \pi'^{-1}_i$. In both $\mathcal{H}$ and $\mathcal{R}$, $View_{x_{\overline{T}}}(\{P_i | i \in \overline{T}\}) = \sigma_{i,j}, i, j \in \overline{T}$ 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 $\pi'_i = \tau_i \circ \sigma^{-1}_{i0} \circ \pi_i$ which is valid due to the closure property.

□

Lemma 29. *A complete CPSS functionality has a strongly secure protocol.*

Proof. We know that the UNIMPC protocol is semi honest secure for CPSS (proved in Section 5.1). Using Lemma 28 we know that it is UC secure for complete CPSS functionalities. Hence every complete CPSS functionality has a strongly secure protocol. □

In the paper [10], they describe a UC secure protocol for disseminated-OR functionality. We use it to show that all disseminated functionalities are UC secure.

For an n-party (not counting disseminator (D)) disseminated functionality F, let F_i be the functionality table that party i receives output from. Disseminated OR is the functionality in which D's input is a bit string, with each bit corresponding to a party, and each party's output is "their" bit along with the OR of all the bits in the input string. We first construct the n-dimensional functionality table F_c, with each dimension's possible input values taken from $F'_i s$ output values: if the value y appears in F_i , it will be an input value in the corresponding dimension of F_c. Consider all inputs x that can be given to F. If $\exists x$ with $F(x) = (y_1, \ldots, y_n)$, then the entry in F_c at input vector $(y_1, \ldots, y_n)$ will be 'valid'. If no such x exists, that entry will be 'invalid'. Now that we have F_c, we can use the following protocol to compute F. We also define $Inv_{f_S}(y_S) = 1$ iff $\forall y_{\overline{S}}$ $F(y_S, y_{\overline{S}}) =$ 'invalid'. The action of abort is defined as:

- Tell everyone OK or abort
- If OK not received from everyone then abort

General Disseminated Protocol:

- D chooses its input x, computes $F(x)$, and sends $F_i(x)$ to party i
- For each subset S $\subseteq [n]$
  - For each $\{\tilde{y}_i\}, i \in S$ such that $Inv(\{\tilde{y}_i\}, y_{\overline{S}}) = 1$:
    - D runs disseminated OR, setting party i's bit to 0 if $\tilde{y}_i = F_i(x)$, 1 otherwise.
    - If D inputs the wrong bit for party i or the OR is 0, party i aborts.
- Party i outputs $F_i(x)$.

Lemma 30. *Given access to the disseminated OR functionality, General Disseminated UC securely realizes any disseminated functionality.*

Proof. We must construct simulators (S) for all selections of corrupt parties that can accomplish the same goals as a real-world adversary (A). As always, the cases of all or no parties corrupt are trivial. Additionally, the case of only D being honest is essentially trivial: the corrupt receiving parties receive all of the information in the system, and there is no honest party receiving output for them to cause to decide on an incorrect output. Let's first consider the case of a corrupt D, with all other parties honest. Again, the nature of the functionality gives both A and S all the information in the system. Then A's goal is to cause the honest parties to decide on output values that are collectively invalid. It's easy to see that this is impossible: if D has sent an invalid combination of outputs, and always inputs the correct values to disseminated OR, it will be caught when the invalid combination it sent is checked. On the other hand, if D ever inputs incorrect values to disseminated OR, the party whose bit was flipped will catch it. An A that causes its corrupt D to deviate from the protocol in any way can then be simulated by an S that always causes an abort. Now, consider the case where D and at least one other party are corrupt. In this case we give the construction of a simulator. The simulator first runs the protocol. This step leads the simulator to get the value y_H where H stands for honest parties. It can check whether there exists an extension possible to y_H so that $(y_H, y_{\overline{H}})$ is a valid output. If there is none then the simulator instructs the functionality to not send output to any party. If there is an extension possible then the simulator sends $(y_H, y_{\overline{H}})$ to the functionality and instructs the functionality to not send the output to the parties which were aborted during the protocol execution in the first step of the simulation. This is a good simulator because if we fix the randomness tape of the adversary it results in the same parties getting aborted and the same output of honest parties. In the case that y_H has a valid extension the simulator knows which party to abort and instructs the functionality accordingly. The simulator ensures that the honest parties receive the same output as in the real world. In the case of no valid extension, while running the protocol there will be a set picked which will represent the set of honest parties and the protocol will abort and the simulator will instruct the functionality to send the output to none of the parties.

Finally, let's consider the case where D and at least one other party are honest. In this case the simulator gets the value $y_{\overline{H}}$. Then it simulates the first and second round of the protocol

described above.This is a good simulator because when it comes to aborts, if some honest party aborts before last round then all abort in the next round as none would hear an OK from the same honest party. □

Theorem 31. *Every disseminated functionality and complete CPSS aggregating functionality has a strongly secure protocol.*

Proof. We have shown in Lemma 29 that every complete CPSS aggregating functionality has a strongly secure protocol. Also, we note that general disseminated protocol described above is trivially semi-honest secure (Disseminator sends $F_i(x)$ in the first round itself). Since we have shown in Lemma 30 that it is also UC-secure, the result follows. □

Theorem 32. *If a functionality $\mathcal{F}$ has a strongly secure protocol Π, then $\mathcal{F}$ is either a disseminating functionality or a CPS.*

Proof. It has been shown in [11] that if a functionality is UC secure it is either aggregating or disseminating. We have shown in Section 4 that if an aggregating functionality is semi honest secure then it must be CPS functionality. For a functionality to be both UC and semi-honest secure it must be either disseminating or a CPS. □

Chapter 10

Future Work

- CPS is a necessary condition for functionalities which have (UNI)MPC protocols and CPSS is a sufficient condition. Further work can be done to reduce this gap between the necessary and sufficient conditions.
- There is a difference between the necessary and sufficient conditions for semi-honest security and that for strong security. More focus can be given to the difference between CPSS and complete CPSS.
- We do not give an example of any CPSS function that cannot be embedded into a CPS functionality. Hence it leaves open the possibility of the two classes being equivalent.
- Further work can be devoted to find the full characterization of UC-secure general MPC functionalities.
- There is a lack of progress on the problem of passively secure MPC for symmetric functionalities. It would be nice to examine what our positive and negative results imply for the symmetric case.
- Work can be done on the characterization of UNIMPC randomness as UNIMPC can't generate arbitrary randomness unlike NIMPC.

Bibliography

[1] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 387–404. Springer, 2014. URL: `https://doi.org/10.1007/978-3-662-44381-1_22`, `doi:10.1007/978-3-662-44381-1_22`.

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

[3] 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.

[4] Benny Chor and Yuval Ishai. On privacy and partition arguments. *Information and Computation*, 167(1):2–9, 2001.

[5] Deepesh Data, Manoj Prabhakaran, and Vinod Prabhakaran. On the communication complexity of secure computation. CoRR Report 1311.7584 available from `http://arxiv.org`, 2013.

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

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

[8] 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 Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 238–255. Springer, 2009.

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

[10] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2008.

[11] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. *Electronic Colloquium on Computational Complexity (ECCC)*, 15(50), 2008.

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

[13] Wikipedia contributors. Universal composability — Wikipedia, the free encyclopedia, 2017. [Online; accessed 9-May-2018]. URL: `https://en.wikipedia.org/w/index.php?title=Universal_composability&oldid=797043101`.

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