Title
Necessary and Sufficient Conditions for General Interaction Patterns for MPC

Author
Mittal, Manika

Publication Date
2017

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Necessary and Sufficient Conditions for General Interaction Patterns for MPC

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Manika Mittal

2017

ABSTRACT OF THE THESIS

Necessary and Sufficient Conditions for General Interaction Patterns for MPC

by

Manika Mittal
Master of Science in Computer Science
University of California, Los Angeles, 2017
Professor Rafail Ostrovsky, Chair

The standard model of Secure Multi-Party Computation (MPC) allows a set of parties with private inputs to compute some joint function of their inputs. The aim of MPC is to allow this computation without revealing any more information than the output of the function. However, achieving this requires a full interaction among the participating parties, which is not always feasible in real life. On the other hand, in certain restricted interaction patterns it is possible for the adversary to learn the output of the function on honest inputs and *every possible combination* of the corrupted inputs. This motivates the study of different interaction patterns and their properties to better understand and formalize the scenarios under which we can *limit* the amount of information an adversary can derive.

The interaction pattern that different parties participate in, to compute a function, can be modeled as a graph. This thesis studies the different properties of this graph and answers the following question - "What is the minimum sequence of edges that are needed to ensure that an adversary cannot collude with other corrupted parties to learn different outputs of the function?". We analyze different scenarios with different bounds on the number of corrupted and output parties.

The thesis of Manika Mittal is approved.

Raghu Meka

Alexander Sherstov

Rafail Ostrovsky, Committee Chair

University of California, Los Angeles

2017

Table of Contents

List of Figures

Acknowledgments

I would like to sincerely thank my academic advisor Prof. Rafail Ostrovsky for providing invaluable support throughout my Master's program. I also want to thank Prof. Yuval Ishai for his support and valuable discussions, without which this work would not have existed. I am also enormously grateful to my Graduate Student Affairs Officers: Steve Arbuckle and Jeanette Reyes, for their great support and guidance during the process of filing this thesis. At last, I would like to gratefully thank the Computer Science Department of UCLA for providing me with the opportunity to work with great research scholars and learn some invaluable lessons.

CHAPTER 1

Introduction

Secure Multi-Party Computation (MPC) enables a set of parties to compute a function on their inputs in such a way that the parties *only* learn the output of the function and nothing about the inputs of the other parties. Most of the prior work [Yao82, GMW87] in the area of MPC involves protocols with *full interaction* i.e. all the parties involved in MPC must remain online and send messages to all other parties. However, realizing full interaction can be expensive and sometimes even infeasible in real world applications. Computing a function with MPC will require $\Omega(n^2)$ messages (where n is the number of parties) even for functions (like majority) that can be computed in $n-1$ messages. Moreover, it might not be possible for *all* the parties to be online at the same time.

All these reasons motivated the study of *restricted* [HLP11] and *general* [HIJ16] interaction patterns. While [HLP11] studies the properties of secure MPC under specific interaction patterns like a star pattern, [HIJ16] tries to understand the impact of a general interaction pattern on MPC, or more specifically if a function can be securely realized by a protocol that complies with an arbitrary interaction pattern. It was observed that under certain conditions the *traditional* definition of security cannot be achieved and so the notion of "best possible security" was formulated. The aim of this thesis is to formulate the conditions when *traditional* definition of security can be achieved by a general interaction pattern.

A general interaction pattern can be modeled as a graph where the sequence of messages that are sent amongst the parties can be represented as an edge sequence. The edge sequence ends in an output party that computes the function. If $x_1, x_2, ..., x_n$ are the inputs of the n parties that wish to compute a function, then the *traditional* model of MPC enforces that

the corrupted parties *must only* learn the value of function f on a *single* input $(x_1, x_2, ..., x_n)$. However, in certain interaction patterns this cannot be achieved and instead we rely on "best possible security" that captures everything that the adversaries can learn by varying some of the inputs from $(x_1, x_2, ..., x_n)$. The inputs that can be varied by the corrupted parties are called *free inputs*, while all other inputs are *fixed inputs*. While the inputs from honest parties are always fixed, some inputs from corrupted parties can also be made fixed. In particular, if a corrupted party has a message path or a subsequence in the interaction pattern that goes *through* an honest party to the output party, then the input of such a corrupted party can be considered fixed. If the corrupted party has no subsequence that goes through an honest party then the input is considered as a free input.

1.1 Our Results

This work formulates the conditions and the constraints that a general interaction pattern *must* follow in order to achieve the traditional definition of security. It answers the following question : "What is the minimum sequence of edges that are needed to ensure that a general interaction pattern achieves the traditional model of security?".

Contributions of this dissertation can be summarized as follows:

Theorem 1 (Informal)**.** *The two-way chain is an optimal interaction pattern that achieves traditional security when there is a single output party and a single honest party.*

Theorem 2 (Informal)**.** *The number of edges required for n parties, with a single corrupted party and a single output party, to achieve traditional model of MPC security is at least n.*

Theorem 3 (Informal)**.** *The number of edges needed for n parties, with one output party and at most c corrupted parties, to achieve traditional model of MPC is given by :*

$$No.\ of\ Edges\ \geq n + c - 1$$

Theorem 4 (Informal)**.** *The number of edges needed for n parties, with o output parties and*

at most c corrupted parties, that achieve traditional model of MPC is given by :

$$No.\ of\ Edges\ \geq min\{n + (c-1) + (o-1), 2n\}$$

CHAPTER 2

Problem Formulation

This chapter introduces the definitions, terms and terminologies that will be used throughout the dissertation. It formulates the problem and enlists some important properties. In all definitions we let $U = \{u_1, ..., u_n\}$ denote the set of parties that participate in the MPC protocol.

Definition 1 (Interaction Pattern)**.** *An N-message interaction pattern for the set of parties U is specified as a sequence of messages given by,*

$$p = (u_1, e_1, u_2, ..., e_N, u_n)$$

where $\{u_i : 1 \leq i \leq n\} \in U$ are the participating parties, and $e_i : 1 \leq i \leq N$ signifies a message sent from the party before e_i in p to a party after e_i in p. The edge/message e_i in p must *be unique and is given a unique sequence number, i. Note that p is a* ***trail*** *i.e. the nodes in p can be repeated but the edges are unique. It is possible that 2 or more parties send messages to a single node simultaneously, in which case the messages/edges are still given unique (increasing) sequence numbers and the final interaction pattern is always in the ascending order of sequence numbers. In such a case, the pattern can be given by,*

$$p = (u_1, e_1, [u_2, e_2], [u_3, e_3], u_4, e_4, ..., e_N, u_n)$$

Here u_1, u_2, u_3 send messages e_1, e_2, e_3 simultaneously to u_4.

Note that the interaction patterns can be modeled as a graph.

Some examples:

Star interaction pattern*: Following is an example of a star interaction pattern p with*

four parties u_1, u_2, u_3, u_4, where u_4 is the output node.

$$p = (u_1, e_1, [u_2, e_2], [u_3, e_3], u_4)$$

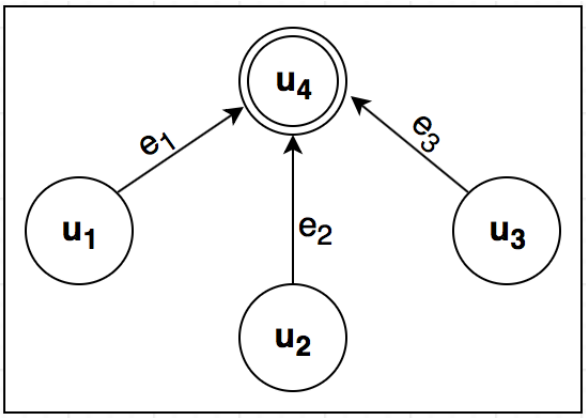


Figure 2.1: Graphical representation of a star interaction pattern

Two-way chain interaction pattern*: Following is an example of a two-way chain interaction pattern p with three parties u_1, u_2, u_3, where u_1 is the output node.*

$$p = (u_1, e_1, u_2, e_2, u_3, e_3, u_2, e_4, u_1)$$

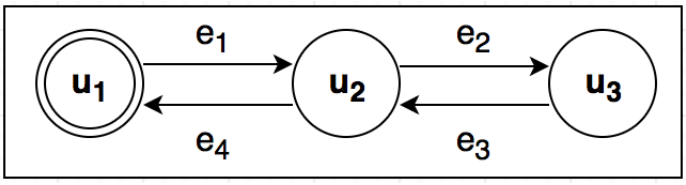


Figure 2.2: Graphical representation of a two-way chain interaction pattern

Definition 2 (Fixed v/s Free Inputs)**.** *Given a set of corrupted parties $C \subset U$, an interaction pattern p and parties $u_i, u_j \in U$ such that u_j is an output party, an input of party u_i is considered fixed if either*

- $u_i \notin C$ *or*
- $\exists$ *a subsequence in* p *from* u_i *to* u_j *through an honest node* $u_h : u_h \notin C$ *with strictly increasing sequence numbers.*

Definition 3 (Traditional Model of MPC v/s Best Possible Security)**.** *In the traditional model of MPC, the corrupted parties can learn no more than the output of the function* f *on* $(x_1, x_2, ..., x_n)$*. Best Possible Security, on the other hand, is defined using residual function that captures everything that the corrupted parties can learn by computing the function* f *on arbitrary choices of free inputs.*

For example, in the star interaction pattern, $p = (u_1, e_1, [u_2, e_2], [u_3, e_3], [u_4, e_4], ..., u_n)$*, if the last* k *parties are corrupted then the adversary can learn the value of the function* $f(x_1, ..., x_{n-k}, x_{n-k+1}, ..., x_n)$ *on fixed* $x_i : 1 \leq i \leq n-k$ *and all possible combinations of the remaining inputs* $x_i : n-k+1 \leq i \leq n$*.*

2.1 Problem Statement

Clearly the goal is to formulate a general interaction pattern that *always* achieves the security required by the traditional model of MPC. In order to ensure that, we need to restrict the value of the residual function to the value of the function f on a *single* input $(x_1, x_2, ..., x_n)$. This can be achieved by ensuring that the inputs from *all* the parties are fixed. The idea is to formulate the conditions of a general interaction pattern in such a way that for every subset of corrupted nodes, the inputs are fixed. More formally,

Given:

- Set of parties/nodes U such that $|U| = n$
- Maximum number of corrupted parties $= c$
- Set of output nodes O such that $|O| = o$

Find the minimum sequence of edges such that:

Condition 1. *For every pair of nodes* $(x, y) : x \in U, y \in O$*, there exists a path (subsequence) from* $x \to y$

Condition 2. *For every subset* $C \subset U : |C| \leq c$ *and every pair of nodes* $(x, y) : x \in C, y \in O \cap C$ *there exists a path (subsequence)* $x \to y$ *going through some node* $h \in U \backslash C$

These conditions ensure that all the parties have fixed inputs and thus achieve the traditional security model of MPC.

An edge sequence is said to be $[n, c, o]$-*valid* if it satisfies the above two conditions.

2.2 Types of Nodes

This sections defines the various types of nodes, used in this thesis, and some of their interesting properties.

2.2.1 Output Node

The output node is the node that computes the joint function and announces/ outputs it. It is denoted by the following symbol:

Figure 2.3: Symbol for Output node

2.2.1.1 Properties

Property 2.2.1.1. *Every output node will have an outdegree of at least 1 and an indegree of at least 1.*

Proof 2.2.1.1. *To satisfy Condition 1 every non-output node must have a path till the output node, thus the output node needs at least one incoming edge. In some subset when the*

output node is corrupt, in order to satisfy Condition 2 the output node must have a path to itself through an honest node, thus every output node will have at least one outgoing edge.

2.2.2 Non-Output Node

If a node is not an output node, it is a non-output node. It is denoted by the following symbol:

Figure 2.4: Symbol for Non-Output node

2.2.2.1 Properties

Property 2.2.2.1. *Every non-output node will have an outdegree of at least 1.*

Proof 2.2.2.1. *To satisfy Condition 1 every non-output node must have a path till the output node, thus it needs at least one outgoing edge.*

Property 2.2.2.2. *For a non-output node, in a* minimal $[n, c, o]$*-valid edge sequence, the maximum of outgoing edge sequence numbers will always be greater than the maximum of incoming edge sequence numbers.*

Proof 2.2.2.2. *If there is an incoming edge (to a non-output node) which has sequence number greater than the maximum of outgoing edge sequence numbers then that means, this incoming edge does not have an outgoing edge with a higher sequence number. Since the node is a non-output node, this incoming edge can be removed without affecting the solution because it serves none of the 2 conditions in the problem statement. But this contradicts our assumption that the* $[n, c, o]$*-valid edge sequence is minimal.*

2.2.3 Hanging Node

A non-output node with NO incoming edges is called a hanging node.

2.2.3.1 Properties

Property 2.2.3.1. *In a* minimal $[n, c, o]$*-valid edge sequence, a hanging node cannot have more than 1 outgoing edges to the same node.*

Proof 2.2.3.1. *More than 1 outgoing edges from a non-output node with no incoming edges to the same node can simply be combined to a single outgoing edge with the highest sequence number. This does not affect the two conditions. But this contradicts our assumption that the the* $[n, c, o]$*-valid edge sequence is minimal.*

2.2.4 Target Node

A non-output node with exactly one outgoing and one incoming edge is called a target node.

2.2.4.1 Properties

Property 2.2.4.1. *A non-output node with outdegree* $= 1$ *and indegree* > 1 *in an* $[n, c, o]$*-valid edge sequence can be transformed to a target node without violating the two conditions.*

Proof 2.2.4.1. *Suppose there is an* $[n, c, o]$*-valid edge sequence that has a non-output node* (u_z) *with outdegree 1 and indegree* $= l > 1$*, then from Property 2.2.2.2, we know*

$$s_{u_z \to any} > max\{s_{u_i \to u_z} : 1 \leq i \leq l\}$$

where $s_{x \to y}$ *denotes the sequence number of the edge/message* $x \to y$ *and* $U' = \{u_i : 1 \leq i \leq l\} \subseteq U$ *denote the set of nodes that have an outgoing edge to* u_z*.*

Transformation: Pick the node $u \in U'$ *that has sequence number* $= max\{s_{u_i \to u_z} : 1 \leq i \leq l\}$*. Remove all the edges* $u_i \to u_z$ *where* $u_i \in U' \backslash u$ *and add edges to* $u_i \to u$ *and keep the same sequence numbers. Since* $s_{u_i \to u_z} < s_{u \to u_z} \quad \forall u_i \in U' \backslash u$*, this modification still satisfies*

the two conditions. In this transformed edge sequence, u_z has an indegree and outdegree of 1.

One example of this transformation, where u_5 is the non-output node with outdegree $= 1$ and indegree > 1, is shown in Figure 2.5.

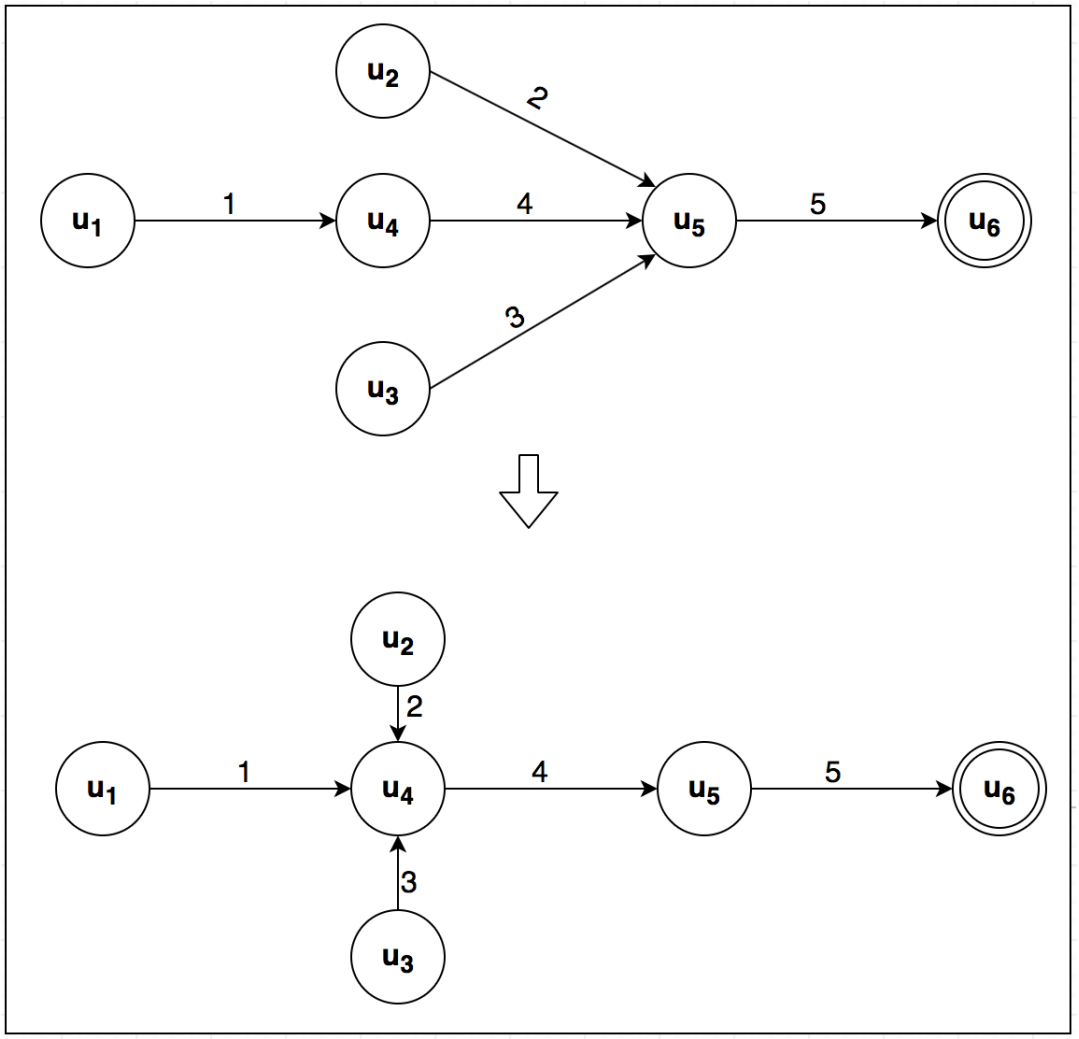


Figure 2.5: An example of Property 2.2.4.1.

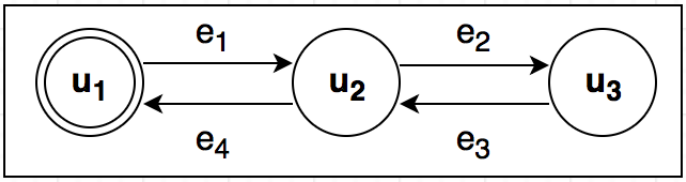
Property 2.2.4.2. *There will be at least $c - 1$ distinct nodes between a target node and the first output node in an $[n, c, o]$-valid edge sequence.*

Proof 2.2.4.2. *Let u_t be a target node. Since u_t has only one incoming and one outgoing edge, it only appears once in the $[n, c, o]$-valid edge sequence, which means that there must*

be $\geq c-1$ distinct nodes (not including u_O) between u_t and u_O, where u_O is the first output node in the sequence. If there are $< c-1$ distinct nodes between u_t and u_O then it violates Condition 2 in the subset where u_t, u_O and the $< c-1$ nodes are corrupt because there will not be any path from u_t to u_O through an honest node. Note that the $c-1$ distinct nodes cannot have u_O as one of the distinct nodes.

CHAPTER 3

Two-Way Chain Interaction Pattern



In this chapter we will show that the two-way chain interaction pattern achieves the traditional model of MPC security when there is a single output node and a single honest node. In particular, we will prove that not only does the two-way chain interaction pattern satisfy the two conditions of the problem statement but it does so with minimal number of edges.

Theorem 5. *The two-way chain forms a minimal* $[n, c = n-1, o = 1]$*-valid edge sequence.*

OR

For $c = n-1$ *and* $o = 1$*, the length of an* $[n, c = n-1, o = 1]$*-valid edge sequence is given by*

$$\textit{Length of } [n, c = n-1, o = 1]\textit{-valid edge sequence } \geq 2n-2$$

Proof. Proof by Induction

Base Case: For $n = 2, c = 1$ and $o = 1$

$$\text{Length of } [n = 2, c = 1, o = 1]\text{-valid edge sequence } \geq 2$$

Let's assume that for $n = 2$ there is a sequence of at most 1 edge satisfying the two conditions. With 1 edge and 2 nodes there are only 2 graphs that are possible that are shown in Figure 3.1 and Figure 3.2. The graph in Figure 3.1 violates Condition 1 because there is no path from

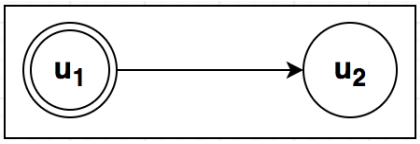


Figure 3.1: Graph 1 with 2 nodes and 1 edge

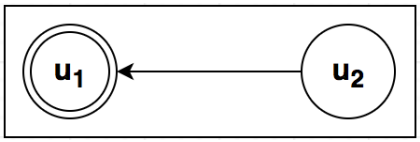


Figure 3.2: Graph 2 with 2 nodes and 1 edge

the non-output node u_2 to the output node u_1. The graph in Figure 3.2 violates Condition 2 in the subset when u_1 is corrupt, since there is no path from the corrupt node u_1 to the output node u_1 through an honest node u_2. Since both the possibilities of 1 edge graphs do not satisfy the two conditions, we know that for $[n = 2, c = 1, o = 1]$-valid edge sequence, we need at least 2 edges.

While the base case can easily be verified by inspection, we prove a more general claim below.

Claim 1. *When $c = n - 1$, every node will have an indegree ≥ 1 and outdegree ≥ 1.*

Proof. *In sets where $|C| = c = n - 1$, there is only 1 honest node. Thus every non-output node needs at least one incoming edge, because it could be the only honest node in set N for some subset of corrupted nodes $|C| = c$. From Property 2.2.1.1 we know that the output node also has at least 1 incoming edge.*

From Properties 2.2.1.1 and 2.2.2.1, we know that every node will have at least 1 outgoing edge.

Inductive Hypothesis:

Suppose for $n = k, c = k - 1$ and $o = 1$

$$\text{Length of } [n = k, c = k - 1, o = 1]\text{-valid edge sequence } \geq 2k - 2$$

Inductive Step:

We need to prove that for $n = k + 1, c = k$ and $o = 1$

$$\text{Length of } [n = k + 1, c = k, o = 1]\text{-valid edge sequence } \geq 2(k + 1) - 2$$

Let's assume that there is a $[n = k + 1, c = k, o = 1]$-valid edge sequence of length at most $2(k + 1) - 3$. Consider such a minimal sequence, and let G be the corresponding graph. We label the edges of G by their order in the given sequence, and refer to these labels as sequence numbers.

We want to prove that if G exists then a better solution exists for $[n = k, n = k-1, o = 1]$ in less than $2k - 2$ edges, contradicting the inductive hypothesis.

Claim 2. *There exists a non-output node $\in G$ with outdegree 1 and indegree 1 which can be removed along with 2 edges, to give a solution for $n = k, c = k - 1$ and $o = 1$ with $2k - 3$ edges*

Proof. *The proof of Claim 2 relies on the following lemma.*

Lemma 1. *There exists at least one target node i.e. one non-output node $\in G$ with outdegree $= 1$ and indegree $= 1$.*

Proof. *Out of $k+1$ nodes, there are k non-output nodes. Each of those k non-output nodes need to have at least one* outgoing edge *from Property 2.2.2.1. Since the total number of outdegrees (i.e. total edges in a directed graph) are $2(k + 1) - 3$, we know that some nodes must have an outdegree more than 1. But at least one non-output node needs to have an outdegree of exactly 1 because $2k > 2(k + 1) - 3 = 2k - 1$.*

By the same argument, we can argue that there exists at least one node with an indegree 1, but these nodes (one with outdegree 1 and one with indegree 1) need not be the same node.

However using Property 2.2.4.1 we can transform a non-output node with outdegree $= 1$ and indegree > 1 to a non-output node with outdegree $= 1$ and indegree $= 1$.

Let the solution i.e. the sequence of edges for graph G that satisfies the 2 conditions for

$[n = k+1, c = k, o = 1]$ *in* $2(k+1)-3$ *edges be denoted by* $p = (u_1, e_1, ..., u_{t-1}, e_{t-1}, u_t, e_t, u_{t+1}, ..., u_o)$, *where* u_t *is the target node and* u_o *is the output node.*

We want to use this graph to create a graph for $[n = k, c = k-1, o = 1]$ *in* $2k-3$ *edges which will contradict our inductive hypothesis.*

Algorithm to get a solution for $[n = k, c = k-1, o = 1]$ by removing a node and 2 edges:

- *If* $u_{t+1} = u_{t-1}$ *then remove* u_t, e_{t-1} *and* e_t.

  Explanation : From Property 2.2.4.2 we know that there are at least $c - 1 = k - 1$ *distinct non-output nodes in* G *between* u_t *and* u_o, *which implies that there are at least* $k - 2$ *distinct non-output nodes between* $u_{t+1} = u_{t-1}$ *and* u_o, *which means that when* u_{t+1} *is corrupt, it will* always *have some honest node path to* u_o *because* $c \leq k - 1$. *So removing* u_t *does not violate condition 2 for any node when* $c = k - 1$.

- *If* $u_{t+1} \neq u_{t-1}$ *then remove* u_t *and* e_t *and all outgoing edges from* u_{t+1} *that have a sequence number less than* $t + 1$.

  Explanation : Remove u_t *and* e_t *and patch the graph by connecting* e_{t-1} *to* u_{t+1} *and the new solution will look like* $p = (u_1, e_1, ..., u_{t-1}, e_{t-1}, u_{t+1}, ..., u_o)$. *Again there are at least* $k-2$ *distinct nodes between* u_{t+1} *and* u_o. *We can remove all the outgoing edges of* u_{t+1} *that have sequence number less than* $t+1$, *because it has an edge* e_{t+1} *using which* u_{t+1} *has a path through* every other $k - 2$ *nodes till it ends in the output node* u_o. *So the edge* e_{t+1} *makes all the other outgoing edges of* u_{t+1} *with sequence number less than* $t+1$ *redundant. And we know that there is at least 1 such outgoing edge with sequence number less than* $t + 1$ *in the original graph* G *because* u_{t+1} *needed a path through* u_t *to* u_o *in the case when* u_t *is the only honest node. And since the* only *outgoing edge of* u_t *was* e_t, u_{t+1} *needs at least one outgoing edge with sequence number* $< t$.

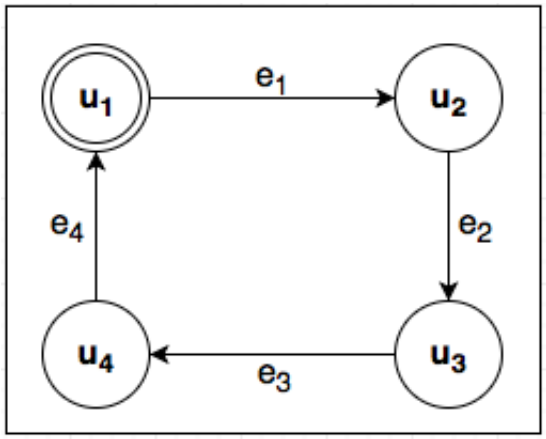
Hence we have proved that if we have a solution for $[n = k+1, c = k, o = 1]$ in $2(k+1)-3$ edges, then we can get a solution for $[n = k, c = k-1, o = 1]$ in $2k-3$ edges which contradicts our inductive hypothesis. Thus we need $\geq 2n - 2$ edges for $[n, c = n - 1, o = 1]$. This

completes the proof by induction. ■

It can be verified that the two-way chain interaction pattern with n nodes has $2n - 2$ edges and is an $[n, c = n - 1, o = 1]$-valid edge sequence. Thus the lower bound in this theorem is tight.

CHAPTER 4

Single Cycle Interaction Pattern



In this chapter we will show that the single cycle interaction pattern achieves the traditional model of MPC security when there is a single output node and a single corrupted node. In particular, we will prove that not only does the single cycle interaction pattern satisfy the two conditions of the problem statement but it does so with minimal number of edges.

Theorem 6. *The single cycle forms a minimal* $[n, c = 1, o = 1]$*-valid edge sequence.*

OR

For $c = 1$ *and* $o = 1$*, the length of an* $[n, c = 1, o = 1]$*-valid edge sequence is given by*

$$\textit{Length of } [n, c = 1, o = 1]\textit{-valid edge sequence } \geq n$$

Proof. From Property 2.2.2.1 and Property 2.2.1.1, we know that every node *must* have *at least* one outgoing edge. Since there are n nodes, this means that we need at least n edges to satisfy the problem conditions.

A single cycle interaction pattern, that starts an ends in the single output node, has exactly n edges and it clearly satisfies the two conditions in the problem statement. Every node has a path/subsequence to the output node and in the subset when the output node is corrupt, the output node has a path to itself through an honest node. In all other subsets, where some non-output node is corrupt, since the output node is honest Condition 2 is always satisfied since $O \cap C = \phi$.

Thus the lower bound mentioned in this theorem is tight. ■

CHAPTER 5

General Interaction Pattern with Single Output Node

In this chapter we will prove a lower bound for the number of edges needed by a general interaction pattern with a single output node to satisfy the two conditions in the problem statement i.e. we will prove a lower bound for the length of an $[n, c, o = 1]$-valid edge sequence.

Theorem 7. *For $o = 1$, the length of an $[n, c, o = 1]$-valid edge sequence is given by*

$$\textit{Length of } [n, c, o = 1]\textit{-valid edge sequence } \geq n + (c - 1)$$

Proof. Consider a *minimal* $[n, c, o = 1]$-valid edge sequence and let G be the corresponding graph. We label the edges of G by their order in the given sequence, and refer to these labels as sequence numbers. The proof depends on the following claims.

Claim 3. *The minimum number of nodes $\in G$ that need at least one incoming edge is given by $max\{c + 1, o\}$*

Proof. *We know that every output node needs at least one incoming edge, thus the number of nodes that have at least one incoming edge is at least o. If $c \geq o$, and if $\leq c$ nodes have incoming edges, then in the subset where all these $\leq c$ nodes are corrupt, there will be no honest node with an incoming edge, violating Condition 2. Thus, we need at least $max\{c + 1, o\}$ nodes to have incoming edges.*

Remark 1. *Since $o = 1$, $c + 1 > o$ is always true.*

Let's assume that G has exactly $c+1$ nodes with incoming edges. The remaining $n-c-1$ nodes are *hanging nodes* and have *at least* a single outgoing edge to satisfy Condition 1.

Consider the subgraph $G' \subseteq G$ that contains just $c+1$ nodes, with the hanging nodes and their corresponding outgoing edges removed. This subgraph must meet the 2 conditions for $[n = c+1, c, o = 1]$ because G meets the 2 conditions in the subset where the $n-c-1$ *hanging* nodes are honest.

$$\begin{aligned} &\#\text{ of edges for } [n = c+1, c, o = 1] \geq 2(c+1) - 2 \text{ (from Theorem 5)} \\ \implies &\#\text{ of edges for } [n, c, o = 1] \geq 2(c+1) - 2 + n - c - 1 \\ \implies &\#\text{ of edges for } [n, c, o = 1] \geq n + c - 1 \\ \therefore \text{ Length of } &[n, c, o = 1]\text{-valid edge sequence } \geq n + (c-1) \end{aligned}$$

From Claim 3 we know that there cannot be a solution for $[n, c, o = 1]$ that has less than $c+1$ nodes with incoming edges.

Now let's assume that G has more than $c+1$ distinct nodes with incoming edges. Let the number of nodes that have at least 1 incoming edge be $k : k > c+1$. The remaining $n-k$ nodes are hanging nodes.

Claim 4. *In the subgraph G', having k nodes (s.t. $k > c+1$) where the hanging nodes are removed, there will always exist a non output node with one incoming and one outgoing edge.*

Proof. *The subgraph is a minimal solution for $[n = k, c, o = 1]$ (The subgraph is a minimal solution because if it wasn't and a better solution existed then we can add the hanging nodes back to this better solution and get a better solution than G which is a contradiction as G is a minimal solution). From the two-way chain proof we know that*

$$\textit{Length of a minimal } [n = k, c, o = 1]\textit{-valid edge sequence } \leq 2k - 2 \tag{5.1}$$

We also know that every node (including the output node) needs at least one outgoing edge. If every non output node has more than 1 outgoing edges then the total number of outgoing edges including the minimum of one outgoing edge for the output node is:

$$\#\textit{ of outgoing edges for } = \#\textit{ of edges for } [n = k, c, o = 1] \geq 2(k-1) + 1$$

But that contradicts Equation 5.1 so there will be at least 1 non output node with a single outgoing edge. And from Property 2.2.4.1 we know that this node will have (or can be transformed to one with) one incoming edge.

Claim 5. *In the subgraph G', having k nodes (s.t. $k > c+1$) where the hanging nodes are removed, if the target node has different parent and child nodes, then it can be transformed to another solution with k nodes with same number of edges but one hanging node.*

Proof. *Let the sequence of edges for G' be given by $p = (..., u_A, e_A, \underline{u_B}, e_B, u_C, ..., u_O)$ where u_O is the output node and u_B is the target node with one incoming edge from u_A and one outgoing edge to u_C .*
From Property 2.2.4.2 we know that there are at least $c-1$ distinct non-output nodes between u_B and u_O.

The edge e_A can be shifted to point to u_C directly instead of u_B while u_B now becomes a hanging node. This shift will not affect the solution, if u_A is a non-output node and has at least $c-1$ distinct non-output nodes between u_A and u_O or if $u_A = u_O$ and has at least c distinct non-output nodes before the sequence ends in u_O. This is to ensure that there is at least one honest node till the path to u_O when u_A is corrupt. All the other nodes remain unaffected by this change.

A more intricate transformation is needed in the case when either there are less than $c-1$ distinct non-output nodes between the non-output u_A and u_O or there are less than c distinct non-output nodes between $u_A = u_O$ and u_O. Since there are at least $c-1$ distinct non-output nodes between u_B and u_O, both these cases happen when there are exactly c nodes after u_B including the final output node u_O and u_A is one of these c nodes.

Since we have $k > c+1$ nodes with incoming edges, we know that there are $k-c-1$ nodes that come before u_A (thus u_A has some incoming edge that comes before e_A). From u_A traverse backwards in the solution sequence till you reach one of the $k-c-1$ nodes, let's call this node u_X. Let e_X be the outgoing edge from u_X, that we just traversed to reach u_X. Shift e_X and make it point to u_C and shift the edge e_A to point to u_X. Reassign e_A and all

the other edges between e_X and e_A a sequence number less than e_X. Since u_X was one of the nodes that did not appear between u_B and u_O, now u_X and u_O have at least $c-1$ distinct non-output nodes between them and all other paths are still valid.

Both these transformations lead to a graph where u_B is now a hanging node.

Remark 2. *When the target node's parent and child nodes are the same, i.e. $u_A = u_C$, then the above transformation will not work. For these cases we use the transformation in Claim 6.*

Claim 6. *In the subgraph G', having k nodes (s.t. $k > c+1$) where the hanging nodes are removed, if the target node has same parent and child nodes, then it can be transformed to a solution for $[k-1, c-1, o=1]$ by removing that node along with its incoming and outgoing edges.*

Proof. *This transformation follows from the algorithm seen in the two-way chain proof. Let the sequence of edges for G' be given $p = (..., u_A, e_A, \underline{u_B}, e_B, u_A, ..., u_O)$ where u_O is the output node and u_B is the target node with parent node = child node = u_A . From Property 2.2.4.2 we know that there are at least $c-1$ distinct non-output nodes between u_B and u_O. Since u_B has incoming and outgoing edge from and to the same node u_A, we know that there are at least $c-2$ distinct non-output nodes between u_A and u_O, which means that if we remove u_B with its incoming and outgoing edge the remaining graph will satisfy the two problem conditions for $[k-1, c-1, o=1]$.*

Algorithm : To count the minimum number of edges in a solution(G) for $[n, c, o=1]$ that has k distinct nodes with incoming edges such that $k > c+1$. We remove the the $n-k$ hanging nodes and pass the subgraph G' with k nodes to the algorithm. Note in this subgraph *all* nodes have at least one incoming edge.

Algorithm 1: To count the number of edges in the solution graph

```
1  Procedure countEdges(subgraph G', k, c)
2      if k = c + 1 then
3          // From Theorem 5
4          return 2(c + 1) − 2 ;
5      else if c == 1 then
6          // From Theorem 6
7          return k ;
8      // From Claim 4 we know that
9      // a target node will always exist
10     else if ∃ a target node with different parent and child then
11         Use Claim 5 to transform G' to G'' with one hanging node ;
12         G' ← G'' with hanging node removed ;
13         return 1 + countEdges(G', k − 1, c) ;
14     else
15         // happens when the target node as same parent
16         // and child nodes
17         Use Claim 6 to transform G' to G'' ;
18         return 2 + countEdges(G'', k − 1, c − 1) ;
19     end
```

Analysis : Let y be the number of times Claim 5 was used and let z be the number of times Claim 6 was used.

Base Case 1 : Suppose the algorithm ended with base case 1 and returned $2(c'+1)-2$. We know that the initial subgraph G' has k nodes where $k > c+1$. Since only Claim 5 reduces the gap between k and $c+1$ until they are equal we know

$$y = k - c - 1$$

Since Claim 5 does not change the value of c and only Claim 6 could have changed it, we know

$$z = c - c'$$

So the total number of edges are at least

$$\begin{aligned}
&\geq \underbrace{n-k}_{\text{initial hanging nodes}} + \underbrace{2(c'+1)-2}_{\text{returned edges}} + \underbrace{y}_{\text{from Claim 5}} + \underbrace{2z}_{\text{from Claim 6}} \\
&\geq n - k + 2(c'+1) - 2 + k - c - 1 + 2(c - c') \\
&\geq n + c - 1
\end{aligned}$$

Base Case 2 : Suppose the algorithm ended with base case 2 and returned k'. We know that the initial subgraph G' has k nodes where $k > c+1$. Since the first base condition was not met, we also know that $k' > 1$. Since only Claim 6 could have reduced the number of corrupted nodes from c to 1, we know

$$z = c - 1$$

Since both the claims reduce the value of k by 1, we know

$$k' = k - y - z$$

$$y = k - k' - z$$

So the total number of edges are at least

$$\begin{aligned}
&\geq \underbrace{n-k}_{\text{initial hanging nodes}} + \underbrace{k'}_{\text{returned edges}} + \underbrace{y}_{\text{from Claim 5}} + \underbrace{2z}_{\text{from Claim 6}} \\
&\geq n - k + k' + k - k' - (c-1) + 2(c-1) \\
&\geq n + c - 1
\end{aligned}$$

■

CHAPTER 6

Generalized Interaction Pattern

In this chapter we will prove a lower bound for the number of edges needed by a general interaction pattern to satisfy the two conditions in the problem statement i.e. we will prove a lower bound for the length of an $[n, c, o]$-valid edge sequence.

Theorem 8. *If* $n + (c-1) + (o-1) < 2n$, *then*

$$\textit{Length of } [n, c, o]\textit{-valid edge sequence } \geq n + (c-1) + (o-1)$$

Proof. In order to prove this we need the following two claims:

Claim 7. *Given a minimal* $[n, c, o]$*-valid edge sequence of length* k *, where* $k < 2n$ *and* $o < n$ *, one can construct an* $[n, c, o+1]$*-valid edge sequence of length* $k+1$*. This* $[n, c, o+1]$*-valid edge sequence is minimal.*

Proof. *Let* p *be a minimal* $[n, c, o]$*-valid edge sequence that meets the two conditions in* k *edges (s.t.* $k < 2n$ *and* $o < n$*) and is given by* $p = (u_1, e_1, ..., e_k, u_{k+1})$*. We know that* u_{k+1} *is an output node (because if it is a non-output node, it can be removed from the sequence along with the edge* e_k *without affecting the two conditions but that contradicts that our solution is minimal).*

Algorithm 1: To construct a $[n, c, o+1]$*-valid edge sequence from a* $[n, c, o]$*-valid edge sequence :*
Pick any non-output node $(u_i : 1 \leq i \leq k)$ *from the* $[n, c, o]$*-valid edge sequence* $p = (u_1, e_1, ..., e_k, u_{k+1})$ *and add an edge from the last node i.e.* u_{k+1} *to this non-output node and make it an output node. The* $[n, c, o+1]$*-valid edge sequence is* $p' = (u_1, e_1, ..., e_k, u_{k+1}, e_{k+1}, u_i)$:

$1 \leq i \leq k$). *which has* $k+1$ *edges.*

Correctness *: Since* p *satisfies the two conditions, every node has a path to the output node* u_{k+1}*. Since in* p'*, the last edge* e_{k+1} *connects* u_{k+1} *to the new output node* u_i*, every node will have a path to this new output node, satisfying Condition 1. Similarly, since for every subset (*$|C| \leq c$*), corrupted nodes have a path till* u_{k+1} *through an honest node, they will also have a path till* u_i*. Thus this new edge sequence* p' *is* $[n, c, o+1]$*-valid.*

Proof of Optimality *: To prove that the new solution is optimal, let's assume that there exists a better edge sequence for* $[n, c, o+1]$ *in* k *edges. Then we can construct an edge sequence for* $[n, c, o]$ *that has* $< k$ *edges which contradicts our initial assumption that* p *is a minimal* $[n, c, o]$*-valid edge sequence.*

Algorithm 2: To construct a $[n, c, o]$*-valid edge sequence from a* $[n, c, o+1]$*-valid edge sequence :*
Let the edge sequence for $[n, c, o+1]$ *be given by* $p' = (u_1, e_1, ..., e_k, u_{k+1})$ *which has* k *edges. Again we know that* u_{k+1} *is an output node. Make this a non-output node. Now we can remove this node from the edge sequence and the corresponding edge, till we reach the second last output node (From Claim 8 below we can see that one will never remove more than one edge). This new edge sequence will have one less output node and will satisfy the conditions in strictly less than* k *edges contradicting that* p *is minimal.*

Claim 8. *In a minimal* $[n, c, o]$*-valid edge sequence, the last* o *nodes will always be output nodes.*

Proof. *Let the minimal* $[n, c, o]$*-valid edge sequence be given by* $p = (u_1, e_1, ..., u_k, e_k, u_{k+1})$*. We know that the two conditions in the problem statement must be satisfied for all the* o *output nodes. Let* u_o *be the first output node to output in the sequence* p*. For* p *to be a solution both the conditions must be satisfied for* u_o*. The claim is that all the nodes after* u_o *in* p *are*

output nodes. Suppose that there is a non output node in the portion of the subsequence from u_o to u_{k+1}, this non-output node and the corresponding edge can be removed because if the conditions are satisfied for u_o, then a direct edge to the next output node is sufficient for the next output node to meet the conditions (like we saw in Algorithm 1 of Claim 7) . But that contradicts our assumption that our sequence of edges is minimal, thus the last o nodes in the solution trail all always output nodes.

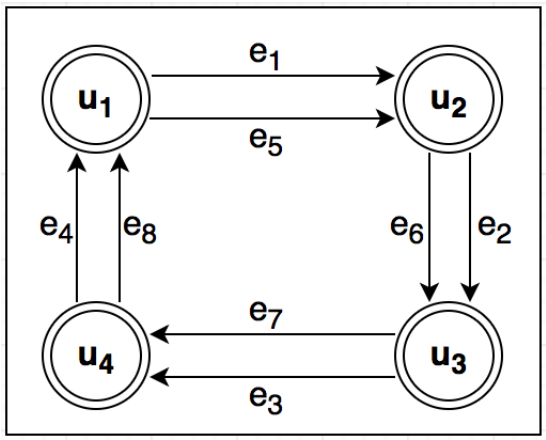
From Algorithm 2 of Claim 7, we know that we can get a minimal $[n, c, o-1]$-valid edge sequence from a minimal $[n, c, o]$-valid edge sequence by changing the last output node in the sequence to a non output node and removing the edges till the second to last output node is reached. From Claim 8 we know that the last o nodes in the minimal $[n, c, o]$-valid edge sequence are output nodes. Thus starting with a minimal $[n, c, o]$-valid edge sequence we can change the last $o-1$ output nodes in the sequence to non-output nodes and remove the last $o-1$ edges to get a minimal $[n, c, o=1]$-valid edge sequence. From Theorem 7, we already know that for a $[n, c, o=1]$-valid edge sequence we need $\geq n + (c-1)$ edges, thus

$$\text{Length of } [n, c, o]\text{-valid edge sequence } \geq n + (c-1) + (o-1)$$

■

CHAPTER 7

Double Cycle Interaction Pattern



In this chapter we will show that a minimal $[n, c, o]$-valid edge sequence can have at most $2n$ edges.

Theorem 9. *For a minimal $[n, c, o]$-valid edge sequence*

$$\textit{Length of a minimal } [n, c, o]\textit{-valid edge sequence } \leq 2n$$

Proof. With $2n$ edges we can make a double cycle graph which will *always* satisfy the two conditions. To prove that it's sufficient to show that the double cycle graph satisfies the two conditions for $[n, c = n - 1, o = n]$, because if a graph satisfies the conditions for $[n, c = n - 1, o = n]$, then it does so $\forall c \leq n - 1$ and $\forall o \leq n$.

For $[n, c = n - 1, o = n]$, we can verify that the double cycle graph satisfies the two conditions by observing the figure above. In a double cycle graph, every output node has a path through every other node before it outputs. And thus, it will satisfy both the conditions. ■

Corollary.

$$\textit{Length of } [n, c, o]\textit{-valid edge sequence } \geq min\{n + (c-1) + (o-1), 2n\}$$

Proof. *From Theorem 8 and Theorem 9.*

CHAPTER 8

Related work

This work is motivated by the work [HIJ16], that studies the impact of a general interaction pattern on MPC. More specifically it answers the question - "Can a function f be securely realized by a protocol that complies with some given interaction pattern?". This work generalizes all interaction patterns, including some common ones like the star interaction pattern, chain, etc. [HIJ16] highlights that full security cannot always be achieved in general interaction patterns. In this thesis, we answer a different question. We look at what conditions a general interaction pattern *must* satisfy in order to ensure full security. After formalizing the conditions, we study what is the minimum number of messages that are needed in a general interaction pattern to satisfy those conditions.

References

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to play any mental game." In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pp. 218–229. ACM, 1987.

[HIJ16] 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*, pp. 157–168. ACM, 2016.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. "Secure computation on the web: Computing without simultaneous interaction." In *Annual Cryptology Conference*, pp. 132–150. Springer, 2011.

[Yao82] Andrew C Yao. "Protocols for secure computations." In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pp. 160–164. IEEE, 1982.